Journal of Computer and System Sciences **57**, 289–308 (1998)
Article No. SS981565

Arity Bounds in First-Order Incremental Evaluation and Definition of Polynomial Time Database Queries*

Guozhu Dong[†]

Department of Computer Science, University of Melbourne, Parkville, Victoria 3052, Australia
E-mail: dong@cs.mu.oz.au

and

Jianwen Su[‡]

Department of Computer Science, University of California, Santa Barbara, California 93106
E-mail: su@cs.ucsb.edu

Received December 6, 1995; revised October 9, 1997

After inserting a tuple into or deleting a tuple from a database, a first-order incremental evaluation system (or a ''foies'') for a database query derives the new query answer by using a first-order query on the new database, the old answer, and perhaps some stored auxiliary relations. Moreover, the auxiliary relations must also be maintained similarly using first-order queries.

In this paper we measure the space needed by foies in terms of the maximal arity of the auxiliary relations and present results on the existence and nonexistence of such space-restricted foies for a variety of graph queries, including counting and transitive closure. We construct space efficient foies for these queries and show that the arity bounds are tight using Ehrenfeucht–Fraïssé games. In particular, for transitive closure over undirected graphs, the minimum arity bound of its foies is shown to be exactly two; this resolves an open problem raised by Patnaik and Immerman in 1994. For the general case, we show that the arity-based hierarchy is strict for all arities. The strictness proof uses queries with input relations having arities much larger than the auxiliary relations. It is still open whether the hierarchy remains strict for arities two or greater when the input relations of queries have arities bounded by a fixed number, such as two, or by the arities of the auxiliary relations.

Finally, we consider a variation of foies where the cost of storing the answer to the query is also ''charged.'' We show that the arity hierarchy in this case is also strict. The positions of queries in the two arity-based hierarchies can differ; we give the positions in this hierarchy of the queries considered for the other arity hierarchy.

1. INTRODUCTION

Recently, there have been research interests focusing on incremental evaluation of database queries (or maintenance of views) [BLT86, AP87, Küc91, WDSY91, DT92, DS93, GMS93, DK97, PI94, RRSS94, DST95, DS95a, DS95b, DLW95, GL95, DP97]. In particular, a framework of evaluating recursive queries (or maintaining recursive views) by incrementally evaluating some nonrecursive, first-order queries has been proposed and studied by Dong, Su, and Topor [DST95, DS95b, DS93]; a slightly different framework was proposed and studied by Patnaik and Immerman [PI94]. In this approach, some auxiliary relations may be computed and stored in addition to the answer to the query of interest; when an update occurs, the new answer to the query and the new auxiliary relations are obtained using some first-order queries (which are fixed for the query of interest) on the new database, the old auxiliary relations, and the old answer. Such an approach is useful for situations where the database changes frequently and the query answer is needed in real time. The approach also has great potential for parallel evaluation; this is because first-order queries are readily adaptable to parallel implementations and have very low parallel complexity [AHV95]. This approach also gives pure relational database systems the power of supporting recursive views without resorting to programming in host languages. It has been shown that many database queries, including regular chain datalog (plus a particular kind of initialization) [DT92, DS95a, DST95], and transitive closure over acyclic graphs [DS95b] and over undirected graphs [PI94], can be maintained in this way.

If a (finite) set of first-order queries maintains the answers to a query (and necessary auxiliary relations), we call the set a first-order incremental evaluation system (or "foies") for the query of interest. In this paper, we consider the space resources that are needed by foies. We measure the space resources in terms of the maximum arity of the auxiliary relations used by a foies: With maximal arity k, the auxiliary relations can hold at most $O(n^k)$ tuples, where n is the number of constants in the input database. We define a

* An extended abstract of the paper appeared under the title "Space-Bounded FOIES" in the "Proceedings of ACM Symposium on Principles of Database Systems, 1995."

† This author gratefully acknowledges support of the Australian research Council through research grants. Part of this work was done while the author was on study leave at UCSB.

‡ Work by this author supported in part by NSF Grant IRI-9109520 and IRI-9411330 and NASA Grant NAGW-3888.

family of classes, FOIES_k, as queries having foies with auxiliary relations of arity $\leqslant k$. We show that the FOIES_k classes are separable at each level. On one hand, we give tight arity bounds for a number of well-known graph queries and show that they separate the FOIES_k hierarchy for the lowest levels. We construct foies with low arity bounds for these queries. In particular, we show that the transitive closure query for undirected graphs is in FOIES_2; this solves an open problem raised by Patnaik and Immerman in [PI94]. We further prove that these queries do not have foies with lower arity bounds using proof techniques based on Ehrenfeucht–Fraïssé games [Fra54, Ehr61]. On the other hand, we show that the arity-based hierarchy FOIES_k is strict for all arities. We define a *k-Mod*-4 query for each $k \in \mathbb{N}$ and show that it is in FOIES_k but not in FOIES_{k-1}. The nonmembership proof is based on a result by Cai [Cai90] which establishes an exponential lower bound on the size of constant-depth Boolean circuits with "help" bits that compute the *multiple parity* problem. In particular, we show that if *k-Mod*-4 is in FOIES_{k-1}, then there is a polynomial size constant-depth Boolean circuit with help bits that computes the *multiple parity* problem, contradicting Cai's result. The input relation of the *k-Mod*-4 query, however, has arity $6k+1$ (improved to $3k+1$ in [DZ97]), much larger than the bound k on auxiliary relations. It remains open whether the classes FOIES_k-and FOIES_{k-1} can be separated by queries over (input) database schema whose arity is k-ary or bounded (such as binary). We note that the negative results reported here are the first of their kind. Finally we discuss what happens to the arity hierarchy if we also charge the cost of storing the answer to the query of interest.

In general, foies can be nondeterministic since the auxiliary relations may be defined by nondeterministic mappings. If the auxiliary relations are defined by deterministic mappings, then the foies is said to be deterministic. In [DS97] we compare the deterministic foies with nondeterministic foies, and we show among other things that nondeterministic foies are more powerful and more space efficient than deterministic ones. The example query (i.e., *k-Mod*-4) used in showing the strictness of the foies arity hierarchy also implies that the deterministic foies arity hierarchy is strict, thus answering our own problem in [DS97].

In addition to related work on view maintenance cited earlier, there are three other groups of related work. (i) The first group is the Dyn-FO framework of [PI94]. There are similarities as well as difference between Dyn-FO and our foies. Both use first-order formulas as maintenance queries. However, Dyn-FO deals with problems which do not seem to correspond to database queries. Furthermore, for one logical update, Dyn-FO allows a first-order query to be evaluated an unbounded number of times (depending on the number of bits affected by the update), whereas in foies each query can only be evaluated once. (ii) The second group generalizes the foies framework in the database setting by using SQL or other query languages, which are more powerful than the first-order query language, as the incremental language [DLW95]. (iii) The third group consists of other kinds of on-line algorithms for graph problems. Concerned with the design of efficient general algorithms instead of with databases, these algorithms typically use nonrelational data structures such as queues or lined lists, use invented values such as integers, or use the iteration constructs such as while loops and for loops (see [DS95b] for references to such work). A more general complexity-theoretical framework for on-line or incremental computation is reported in [MSVT94]. Motivated from the view maintenance problem of databases, our approach uses only relations and has none of the above nonlogical features. A comparison of our approach with other methods for the incremental evaluation of database queries is given in [DST95].

This paper is organized as follows. Section 2 reviews some basic notions including, in particular, foies. Section 3 discusses the classes FOIES_0 and FOIES_1 and their relationships to monadic Σ^1_1. Section 4 focuses on queries in FOIES_2. Section 5 includes the techniques for proving negative results concerning the classes of $\mathrm{FOIES}_{\leqslant 1}$. In Section 6 we prove that the FOIES_k hierarchy is strict for all k. In Section 7 we consider the alternative arity hierarchy. Finally in Section 8 we summarize our results and list some open problems. In the Appendix we prove that same-generation and transitive closure over k-path graphs belong to FOIES_4.

2. THE NOTION OF FOIES

In this section, we provide necessary terminology about databases and graphs, and define the central notion of a first-order incremental evaluation system of a database query. We illustrate first-order incremental evaluation systems with a few examples.

Roughly speaking, a first-order incremental evaluation system is simply a finite set of first-order queries which maintains a set of derived relations, including the answer to a query of interest, after base relations are updated.

Formally, we assume the existence of two disjoint infinite sets: a set **rel** of *predicate* or *relation names*, each of which has an associated *arity* $n \geqslant 0$, and a (universal) domain **dom** of *constants*.

Let $\mathbb{N}$ denote the set of natural numbers. For each $n \in \mathbb{N}$, an *n-ary tuple* is a mapping from $\{i \mid 1 \leqslant i \leqslant n\}$ to **dom**, and an *n-ary relation* is a finite set of n-ary tuples. Observe that there is exactly one 0-ary tuple, which is denoted as $[\ ]$. The *cardinality* of a relation r, denoted by $|r|$, is the number of tuples in r. A *database schema* is a finite subset of **rel**, and its *arity* is the maximal arity of relation names contained in it. A *database instance* of a database schema $\mathscr{S}$ is a mapping I such that, for each relation name R in $\mathscr{S}$, if R has arity n, then $I(R)$ is an n-ary relation; the *cardinality* of I, denoted by $|I|$, is defined as the total number of tuples in I, i.e.,

$|I| = \sum_{R \in \mathcal{S}} |I(R)|$. We also denote by *inst*($\mathcal{S}$) the set of all database instances of $\mathcal{S}$.

In this paper we are interested in the maintenance, in first order, of answers to queries after changes are made to base relations. To formalize this approach, we need to introduce several necessary concepts, including "queries," "auxiliary stored database definitions," "input domain preserving," and "permissible updates."

Let $k \in \mathbb{N}$, $\mathcal{S}_{\text{in}}$ be a database schema, and R_Q be a k-ary relation name not in $\mathcal{S}_{\text{in}}$. A *k-ary query from $\mathcal{S}_{\text{in}}$ to R_Q* is a (possibly partial) mapping Q from *inst*($\mathcal{S}_{\text{in}}$) to relation instances of R_Q satisfying the following "genericity" criterion [CH82, Hul86]: For each permutation ρ of **dom**, Q commutes with ρ (ρ is extended naturally to relations and databases), i.e., $\forall I \in \mathit{inst}(\mathcal{S}_{\text{in}})$, $Q(\rho(I)) = \rho(Q(I))$. We will refer to relation names in $\mathcal{S}_{\text{in}}$ (and their instances) as *base* relations for Q.

We note that a query Q is always deterministic but it may be partial; i.e., for each database instance I, $Q(I)$ is unique if it is defined. Practically, partial queries can be viewed as queries over databases with integrity constraints. We call the family of database instances over which Q is defined the *domain* of Q and denote it by *dom*(Q).

First-order queries are formulas built using $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\exists$, and $\forall$, starting from *atoms* of the form $R(t_1, ..., t_m)$ and $t_1 = t_2$, where each t_i is either a variable or a constant and R is a relation name of arity m. The *arity* of a first-order formula (query) is the number of free variables in the formula.

Suppose $\mathcal{S}_{\text{aux}}$ is a database schema disjoint from $\mathcal{S}_{\text{in}}$. We define an *auxiliary stored database definition* (*aux-d-def*) from $\mathcal{S}_{\text{in}}$ to $\mathcal{S}_{\text{aux}}$ to be a (possibly partial) mapping from *inst*($\mathcal{S}_{\text{in}}$) to finite subsets of *inst*($\mathcal{S}_{\text{aux}}$). Intuitively, aux-d-defs are used to define some stored "intermediate" relations including the query answer. The intermediate relations are then used to help derive new intermediate relations after base relations are updated. Observe than an aux-d-def α may be nondeterministic: For each database instance $I \in \mathit{inst}(\mathcal{S}_{\text{in}})$, every database in $\alpha(I)$ is a possible result (to be stored).

In this paper we focus on the size of such intermediate relations. To relate the size of the intermediate relations with the size of the input relations, we restrict aux-d-defs to be "input domain preserving." Basically this condition says that the auxiliary relations can use at most a bounded number of constants not occurring in the base relations at any time. Let r be a k-ary relation ($k \in \mathbb{N}$). We define the *active domain* of r, denoted by *adom*(r), to be the set of constants occurring in r, i.e., $\mathit{adom}(r) = \bigcup_{1 \leqslant i \leqslant k} \pi_i(r)$, where $\pi_i(r)$ ($1 \leqslant i \leqslant k$) is the projection of r onto the ith column. For each database instance I of a schema $\mathcal{S}$, we define the *active domain* of I to be $\mathit{adom}(I) = \bigcup_{R \in \mathcal{S}} \mathit{adom}(I(R))$. Finally, for each finite set $J = \{r_1, ..., r_k\}$ of relations, we define the *active domain* of J to be $\mathit{adom}(J) = \bigcup_{1 \leqslant i \leqslant k} \mathit{adom}(r_i)$.

An aux-d-def α from $\mathcal{S}_{\text{in}}$ is said to be *input domain preserving with respect to C*, where C is a set of constants, if for each $I \in \mathit{inst}(\mathcal{S}_{\text{in}})$, $\mathit{adom}(\alpha(I)) \subseteq \mathit{adom}(I) \cup C$. Moreover, α is said to be *input domain preserving* if it is input domain preserving with respect to some finite C.

Each database update considered here may insert tuples to and delete tuples from a database. Given a database schema $\mathcal{S}_{\text{in}}$ an *update over* $\mathcal{S}_{\text{in}}$ is a pair $\delta = (\triangle, \nabla)$, where $\triangle, \nabla \in \mathit{inst}(\mathcal{S}_{\text{in}})$. In an update $\delta = (\triangle, \nabla)$, $\triangle$ contains the set of tuples to be inserted, and ∇ the set of tuples to be deleted. For technical convenience, we restrict ourselves to updates whose insertion set and deletion set are disjoint; i.e., no tuple is inserted and deleted by the same update. Each update $\delta = (\triangle, \nabla)$ defines a mapping from *inst*($\mathcal{S}_{\text{in}}$) to *inst*($\mathcal{S}_{\text{in}}$) in the natural way:

- For each $I \in \mathit{inst}(\mathcal{S}_{\text{in}})$, the new instance, denoted by δI, is defined such that $\delta I(R) = I(R) \cup \triangle(R) - \nabla(R)$ for each $R \in \mathcal{S}_{\text{in}}$.

Given a query Q, a first-order incremental evaluation system for Q will maintain the answer to Q as long as the updates are "permissible." Intuitively, the notion of a permissible update is to allow only "small" updates which map *dom*(Q) to *dom*(Q). We require updates to be small and having the databases in the domain of Q all the time, since otherwise there cannot be any first-order queries to maintain any truly recursive query.

Given a database instance $I \in \mathit{dom}(Q)$, a *semi-permissible update* is an update δ such that $\delta I \in \mathit{dom}(Q)$. When Q is given, there is usually a small number ℓ_Q such that each semi-permissible update δ is equivalent to the composition of a (finite) sequence of semi-permissible updates $\delta_1 \cdots \delta_n$ where each $\delta_i = (\triangle_i, \nabla_i)$ and $|\triangle_i| + |\nabla_i| \leqslant \ell_Q$. Because of this, we define an update $\delta = (\triangle, \nabla)$ to be *permissible* if it is semi-permissible and $|\triangle| + |\nabla| \leqslant \ell_Q$. Since it is usually clear from the context what ℓ_Q is, we will not always spell out what ℓ_Q is.

EXAMPLE 2.1. Let $\mathcal{S}_{\text{set}}$ be a database schema consisting of a single unary relation R. The Boolean (0-ary) query *Even* over $\mathcal{S}_{\text{set}}$ is defined, for each database $I \in \mathit{inst}(\mathcal{S}_{\text{set}})$, by $\mathit{Even}(I) = \mathit{true}$ (represented by $\{[\ ]\}$) if the relation $I(R)$ contains an even number of constants, and $\mathit{Even}(I) = \mathit{false}$ (represented by the empty set $\varnothing$) otherwise. So *dom*(*Even*) is the set of all possible databases: $\mathit{dom}(\mathit{Even}) = \mathit{inst}(\mathcal{S}_{\text{set}})$; consequently every update over $\mathcal{S}_{\text{set}}$ is semi-permissible. Each permissible update either inserts exactly one tuple or deletes exactly one tuple (but not both).

Let $\mathcal{S}_{\text{cyc}}$ be a database schema with a single binary relation G. Consider a query over $\mathcal{S}_{\text{cyc}}$ which is defined only when G (review as a finite graph) is a simple cycle. Obviously, each permissible update either replaces one edge of the form (a, b) by two edges (a, c) and (c, d), where c is a new constant, or replaces the two edges (a, c) and (c, b) by (a, b).

We now define a first-order incremental evaluation system for a query. Intuitively, such a system consists of a finite set of first-order queries (see Fig. 1): Each has as input the old database, the old auxiliary relations, the old answer to the query, and the update; for each permissible update, the new auxiliary relations and the new query answer are obtained by the first-order queries. For simplicity, for each database instance (or relation name) X, we will use X^o to refer to the instance *before* an update, and X^n the instance *after* the update.

DEFINITION 2.2. Let Q be a query from $\mathscr{S}_{\text{in}}$ to R_Q and $k \in \mathbb{N}$. A *k-ary first-order incremental evaluation system* (foies) for Q is a triple $(\mathscr{S}_{\text{aux}}, \alpha, \mathscr{A}^{\pm})$ satisfying the following conditions:

1. $\mathscr{S}_{\text{aux}}$ is a k-ary database schema disjoint from $\mathscr{S}_{\text{in}} \cup \{R_Q\}$;
2. α is an input domain preserving aux-d-def from $\mathscr{S}_{\text{in}}$ to $\mathscr{S}_{\text{aux}} \cup \{R_Q\}$ such that $J(R_Q) = Q(I)$ for each $I \in inst(\mathscr{S}_{\text{in}})$ and $J \in \alpha(I)$;
3. $\mathscr{A}^{\pm}$ maps each relation name R in $\mathscr{S}_{\text{aux}} \cup \{R_Q\}$ to a first-order query $\mathscr{A}^{\pm}_R$ (which is called an *aux-maintaining* query); and
4. For each database I^o_{in} in $dom(Q)$, each permissible update $\delta = (\triangle, \nabla)$, and reachable auxiliary database[1] I^o_{Qaux} in $\alpha(I^o_{\text{in}})$, if $I^n_{\text{in}} = \delta I^o_{\text{in}}$ is the new database, then a new auxiliary database I^n_{Qaux} in $\alpha(I^n_{\text{in}})$ is obtained by

$$I^n_{\text{Qaux}}(R) = \mathscr{A}^{\pm}_R(I^o_{\text{in}}, I^o_{\text{Qaux}}, \triangle, \nabla) \qquad \text{for each} \quad R \in \mathscr{S}_{\text{aux}} \cup \{R_Q\}.$$

A (k-ary) foies is *space free* if $\mathscr{S}_{\text{aux}} = \varnothing$. An *insertion-only* foies can only handle insertions.

Observe that we used the subscript "Qaux" to indicate that I_{Qaux} is an instance over $\mathscr{S}_{\text{aux}} \cup \{R_Q\}$. In Fig. 1, $\mathscr{S}_{\text{aux}} \cup \{R_Q\}$ is the set of relations $r_1, ..., r_m$, and $\mathscr{A}^{\pm}_{r_i} = \alpha^{\pm}_i$.

We now illustrate the notion with a very simple example where only the query answer is stored; more examples will be given in Sections 3 and 4.

EXAMPLE 2.3. The query *Even* over the schema $\mathscr{S}_{\text{set}}$ (cf. Example 2.1) has a space-free foies $(\varnothing, Even, \mathscr{A}^{\pm})$, where $\mathscr{A}^{\pm}_{Even}$ is constructed as follows.

If an update inserts a constant e ($\triangle = \{R(e)\}$), the new answer $Even(I^n)$ is *true* iff (i) e was not in the old R and the old answer was *false*, or (ii) e was already in the old R and the old answer was *true*; so,

$$Even(I^n) = \mathscr{A}^{\pm}_{Even}(I^o, Even(I^o), \{e\}, \varnothing) = true$$

[1] We say that the auxiliary database $I^o_{\text{Qaux}} \in inst(\mathscr{S}_{\text{aux}} \cup \{R_Q\})$ is *reachable* for I^o_{in} if there exists a sequence $\delta_n \cdots \delta_1$ of permissible updates such that $\delta_n \cdots \delta_1 \varnothing = I^o_{\text{in}}$ and I^o_{Qaux} is the final auxiliary database obtained by successively applying $\alpha^{\pm}$ after each of these updates.

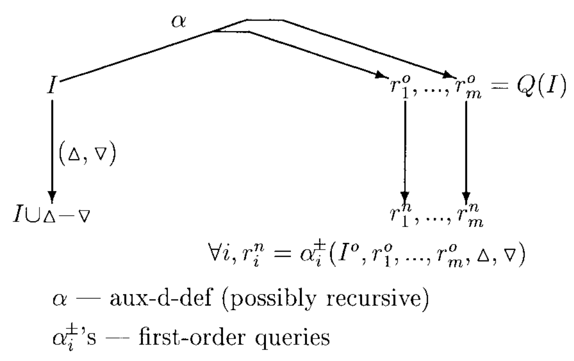

α
I
$r_1^o, ..., r_m^o = Q(I)$
$(\triangle, \nabla)$
$I \cup \triangle - \nabla$
$r_1^n, ..., r_m^n$
$\forall i, r_i^n = \alpha_i^{\pm}(I^o, r_1^o, ..., r_m^o, \triangle, \nabla)$
α — aux-d-def (possibly recursive)
$\alpha_i^{\pm}$'s — first-order queries

FIG. 1. Illustration of a foies for a query Q.

iff

$$(\neg R^o(e) \wedge Even(I^o) = false) \vee (R^o(e) \wedge Even(I^o) = true).$$

Similarly, if an update deletes e ($\nabla = \{R(e)\}$), $Even(I^n)$ is *true* iff (i) e was in the old R and the old answer was *false*, or (ii) e was not in the old R and the old answer was *true*; so,

$$Even(I^n) = \mathscr{A}^{\pm}_{Even}(I^o, Even(I^o), \varnothing, \{e\}) = true$$

iff

$$(R^o(e) \wedge Even(I^o) = false) \vee (\neg R^o(e) \wedge Even(I^o) = true).$$

In the above definition, we did not specify how the old answer and the old auxiliary relations are initialized. There are two options: (i) They are initialized by other means, e.g., by evaluating Q and the aux-d-def on the initial database. (ii) A foies starts from the empty database $\varnothing$. Once initialized, the foies derives the new auxiliary databases by computing its first-order aux-maintaining queries.

The requirement that the auxiliary stored database definitions be input domain preserving is to prevent them from using arbitrarily many constants to encore information, i.e., "cheating." For example, this requirement disallows aux-maintaining queries that do not remove constants from the auxiliary relations even after all tuples containing these constants have been removed from the base relations. As we mentioned earlier, this requirement is important for the study of the space usage (complexity) of foies discussed below.

We measure the space resources used by a foies in terms of the maximum arity of the stored auxiliary relations: With maximal arity k, the auxiliary relations can hold at most $O(n^k)$ tuples, where n is the number of constants in the input database. Thus this arity is an indicator of how hard it is to maintain the query using foies.

There are two options for defining the arity requirement of a first-order incremental evaluation system, depending on whether the arity of the answer relation is included. We will first consider the case where the arity of the answer relation is excluded, and we will discuss what happens for the other option in Section 7.

DEFINITION 2.4. For each $k \in \mathbb{N}$, let FOIES_k be the class of queries having k-ary foies; let FOIES_{sf} be the class of queries having space-free foies; and let $\text{FOIES} = \text{FOIES}_{\text{sf}} \cup \bigcup_{k \in \mathbb{N}} \text{FOIES}_k$.

Obviously, $\forall k \in \mathbb{N}$, $\text{FOIES}_{\text{sf}} \subseteq \text{FOIES}_k \subseteq \text{FOIES}_{k+1}$. The following closure properties of the FOIES_k classes are easily verified.

PROPOSITION 2.5. 1. *FOIES is closed under first-order operations (projection, selection, cross product, set operations, and complement).*

2. *If Q_i $(i = 1, 2)$ is an ℓ_i-ary query in FOIES_{k_i}, then the projection and selection of Q_i are in $\text{FOIES}_{\max\{k_i, \ell_i\}}$; the difference, union, and cross product of Q_1, Q_2 are in $\text{FOIES}_{\max\{k_1, k_2, \ell_1, \ell_2\}}$.*

3. *For each $x \in \{\text{sf}\} \cup \mathbb{N}$, FOIES_x is closed under complement.*

Proof. For (1), let $\mathscr{F}_1$, $\mathscr{F}_2$ be two foies for queries Q_1, Q_2 (respectively). Consider a first-order (binary) operation θ. We can easily combine $\mathscr{F}_1$, $\mathscr{F}_2$ into a foies for $Q_1 \theta Q_2$ by retaining all auxiliary mappings in $\mathscr{F}_1$, $\mathscr{F}_2$ including the answers to Q_1, Q_2 and constructing easily the aux-maintaining query for $Q_1 \theta Q_2$. Unary operations are similar. Item (2) follows from this construction. To see (3), we note that the answer to a query Q can be computed in first order from its complement (with respect to the active domain). ∎

It is interesting to compare FOIES with Σ^1_1. Observe that each query in FOIES is in PTIME. Since (monadic) Σ^1_1 contains NPTIME-complete problems (e.g., 3-colorability), $\text{FOIES} \neq \Sigma^1_1$ unless PTIME = NPTIME. The following is also known: If a Boolean query Q over k-ary input relations has a foies, then Q is in $(k+1)$-ary Σ^1_1 [DW97].

Some of the example queries discussed below are defined over graphs. We now introduce some basic definitions concerning graphs in brief. A graph is represented by a binary relation. We will consider as synonyms these three pairs of terms: a "binary relation" and a "graph," a "binary tuple" and an "edge," and a "constant" and a "node."

Suppose G is a graph. A sequence $u_0 u_1 \cdots u_n$ $(n > 0)$ of nodes in G is a *walk* (from u_0 to u_n) if (u_{i-1}, u_i) is in G for each $1 \leqslant i \leqslant n$; the sequence if a *path* if it is a walk and $u_i \neq u_j$ whenever $0 \leqslant i < j \leqslant n$ such that $i \neq 0$ or $j \neq n$; and the sequence is a *cycle* if it is a path and $u_0 = u_n$. G is *acyclic* if it contains no cycles.

The transitive closure of a graph G, denoted by TC_G, is defined as $\{(x, y) \mid \text{there exists a path of positive length from } x \text{ to } y \text{ in } G\}$. Notice that edges of the form (u, u) contribute only in a trivial way to transitive closures. Due to this reason, we assume that there are no such edges whenever the discussion is on transitive closure. In the remainder of this paper, we let $\hat{R} = R \cup \{(v, v) \mid v \text{ occurs in } R\}$ for each binary relation R.

For each positive integer k, a graph is a *k-path graph* if for each ordered pair of (not necessarily distinct) nodes u, v, there are at most k paths from u to v. In [DS95b], 1-path graphs were referred to as 0-1-path graphs; the singly connected (also called uni-connected) graphs [BC93] are strictly more general class: a graph is singly connected if for each ordered pair of distinct nodes u, v, there is at most one path from u to v.

3. FOIES$_{\leqslant 1}$

In this section we present several queries and their foies to further illustrate the notion of foies. We also state the results that the two containments $\text{FOIES}_{\text{sf}} \subseteq \text{FOIES}_0 \subseteq \text{FOIES}_1$ are proper; the membership proofs are presented here, while the nonmembership proofs will be provided in Section 5. Finally, we compare these three classes with monadic Σ^1_1.

We start with foies for the transitive closure queries. The first is over arbitrary directed graphs but we restrict updates to insertions only, whereas the second is over acyclic directed graphs and both insertions and deletions are allowed. In what follows we shall use the queries in these foies frequently.

EXAMPLE 3.1. We first consider the transitive closure query and restrict the updates to insertions. The query has an insertion-only space-free foies, where for each $e = (u, v)$ and $G = G^o \cup \{e\}$, TC_G is computed by (we also treat a query as a predicate)

$$TC_G(x, y) = TC_{G^o}(x, y) \vee \widehat{TC}_{G^o}(x, u) \wedge \widehat{TC}_{G^o}(v, y).$$

The above provides an efficient way of updating (computing) the transitive closure query; more general results for insertion-only foies can be found in [DT92, DS93, DST95].

EXAMPLE 3.2. Consider now the transitive closure query defined over *acyclic directed graphs*. It has a space-free foies for both insertions and deletions. Insertions are dealt with as in Example 3.1. For deletions, suppose G^o is an acyclic graph, (a, b) an edge in G^o, and $G = G^o - \{(a, b)\}$. If $(x, y) \in TC_{G^o}$, then (x, y) remains in TC_G iff (proved in [DS95b]) $(\phi_0 \to (\phi_1 \vee \phi_2 \vee \phi_3)) \vee \psi$, where

1. ϕ_0 says that x, y, a, b are distinct nodes such that $TC_{G^o}(x, a) \wedge TC_{G^o}(b, y)$.

2. ϕ_1 says there exists a node u on a path from x to y in G^o such that either $\neg\widehat{TC}_{G^o}(u, a) \wedge \neg\widehat{TC}_{G^o}(a, u)$ or $\neg\widehat{TC}_{G^o}(u, b) \wedge \neg\widehat{TC}_{G^o}(b, u)$.

3. ϕ_2 says there exists a node u such that $TC_{G^o}(a, u) \wedge TC_{G^o}(u, b)$. (Note that this implies $u \notin \{a, b\}$.)

4. ϕ_3 says there exist two nodes u and v such that $\widehat{TC}_{G^o}(x, u) \wedge \widehat{TC}_{G^o}(u, a) \wedge \widehat{TC}_{G^o}(b, v) \wedge \widehat{TC}_{G^o}(v, y) \wedge G^o(u, v)$ and either $u \neq a$ or $v \neq b$.

5. ψ handles the case where $x = a$ or $y = b$ in a manner similar to the above.

The following two queries are in FOIES_0 and FOIES_1 (respectively) but not in lower classes.

EXAMPLE 3.3. Consider the Boolean query *Mod-3*, defined on databases over the schema $\mathcal{S}_{\text{set}}$ consisting of one unary relation. It answers *true* if the cardinality of the relation is a multiple of 3 and *false* otherwise. The query is in FOIES_0: We use three 0-arity auxiliary relations p_i ($0 \leqslant i \leqslant 2$), where p_i is nonempty iff there are n constants in the input such that $i = n \bmod 3$. The first-order aux-maintaining queries are easily constructed.

EXAMPLE 3.4. The Boolean query *Equal-Length-Chain* over two binary relations (q_1 and q_2) is defined if each q_i ($i = 1, 2$) contains a single chain; the answer is *true* iff the two chains are of the same length. Then *Equal-Length-Chain* is in FOIES_1. Indeed, each permissible update must map a chain to a chain, so it either adds an edge to or deletes an edge from either the head or the tail of a chain. We use two unary auxiliary relations, p_1 and p_2. Each p_i is to hold the extra nodes in q_i not paired up with nodes in the other chain, i.e., nodes in q_i beyond its prefix of length $\min(|q_1|, |q_2|)$. Clearly, the answer is *true* iff $p_1 = p_2 = \emptyset$.

We update p_1 and p_2 by using the order implied by the chains. For example, for the database shown in Fig. 2, if we insert into q_2 the tuple (2, 3), we simply remove d, the first element of p_1^o, from p_1^o. Other updates can be dealt with similarly.

We now describe the incremental queries for maintaining p_1 and p_2 in English; they can be easily translated into first-order queries. Consider an insertion of an edge (a, b) into q_1^o, the other updates being similar. Then either b is the first node in q_1^o or a is the last node in q_1^o. Consider the first case, the second case is similar and simpler. Suppose $q_1^o \neq \emptyset$.

(i) If $p_1^o = p_2^o = \emptyset$, we insert the last node of q_1^o into p_1.

(ii) If $p_1^o = \emptyset$ but $p_2^o \neq \emptyset$, then p_2 is obtained by removing the first node of q_2 in p_2^o from p_2^o.

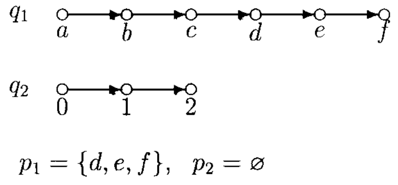


FIG. 2. Two chains and the auxiliary relations.

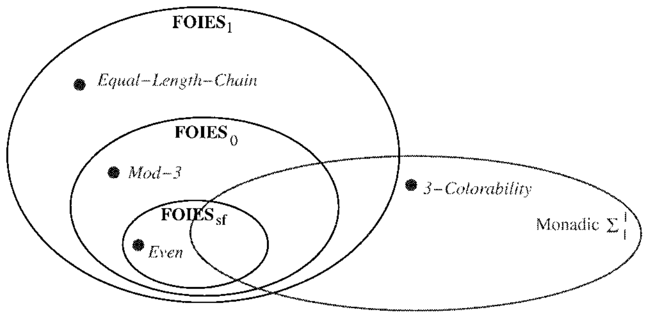


FIG. 3. $\text{FOIES}_{\leqslant 1}$ classes and monadic Σ_1^1.

(iii) If $p_1^o \neq \emptyset$ but $p_2^o = \emptyset$, then p_1 is obtained by adding to p_1^o the predecessor of the first node of q_1 in p_1^o.

(iv) It is not possible to have both $p_1^o \neq \emptyset$ and $p_2^o \neq \emptyset$.

For $q_1^o = \emptyset$ (the first edge introduces two new nodes), we modify the updates on p_1 and p_2 so that two elements are added for (i) and removed for (ii).

THEOREM 3.5. *The* FOIES_k *hierarchy is strict for* $k \leqslant 1$: $\text{FOIES}_{\text{sf}} \subsetneq \text{FOIES}_0 \subsetneq \text{FOIES}_1$.

Proof. The containments clearly hold. The first inequality, $\text{FOIES}_{\text{sf}} \neq \text{FOIES}_0$, holds since *Mod-3* belongs to FOIES_0 as shown in Example 3.3 but it is not in FOIES_{sf}, proved in Theorem 5.3.

For the second inequality, $\text{FOIES}_0 \neq \text{FOIES}_1$, we use the query *Equal-Length-Chain* which is in FOIES_1 as shown in Example 3.4. The result that *Equal-Length-Chain* $\notin \text{FOIES}_0$ will be established in Theorem 5.4 using Ehrenfeucht–Fraïssé games. ∎

It is interesting to compare the $\text{FOIES}_{\leqslant 1}$ classes with monadic Σ_1^1. On one hand, the query *Even* is in FOIES_{sf} (Example 2.3) but not in monadic Σ_1^1 [Fag75, AF90, FSV95]. On the other hand, disconnectivity of undirected graphs is in Σ_1^1 [AF90] but not in FOIES_1 and not in FOIES_{sf} (Theorem 5.7). Therefore, monadic Σ_1^1 is incomparable with $\text{FOIES}_{\leqslant 1}$ classes and with FOIES_{sf} (Fig. 3).

4. FOIES_1 VS FOIES_2 (UNARY VS BINARY)

Patnaik and Immerman [PI94] showed that transitive closure of undirected graphs can be maintained in first order with *ternary* auxiliary relations and posed the open problem of whether *binary* auxiliary relations suffice. In this section we resolve this open problem by giving a positive answer; we will show in Section 5 that this space bound is also tight (unary auxiliary relations are not enough). So FOIES_2 strictly contains FOIES_1. We also show that a number of other conventional queries are in FOIES_2 but not in FOIES_1.

In resolving the open problem, the key idea in the construction is to store a *directed* "spanning forest," instead of the undirected spanning forest as in [PI94], for each undirected graph. This idea is applied to other problems (minimum spanning trees and 2-colorability) on undirected graphs. We also use a "core" of a directed 1-path graphs in assisting the derivation of transitive closure, where the core is an arbitrary maximal acyclic subgraph; this also improves an earlier space upper bound [DS95b] to the optimum.

Our results on undirected graphs and the lack of similar results for directed graphs support, if not confirm, the belief that directed graphs are harder than undirected graphs [AF90]. For maintaining transitive closures of general directed graphs, the above two kinds of "generators" do not seem to help.

We will represent undirected graphs as symmetric binary relations. (The proofs for our results would be much easier if an asymmetric representation were used.) Thus each permissible update (either an insertion or a deletion) should consist of two ordered pairs of the form $\{(x, y), (y, x)\}$. To simplify the notations, we will use an undirected edge $\langle x, y\rangle$ to denote the set of directed edges (x, y) and (y, x), and we will say (x, y) and (y, x) are the two directions of $\langle x, y\rangle$. We say an undirected edge $\langle x, y\rangle$ is *isolated* in a graph if both x and y are connected only to themselves.

To establish the main result of this section, we will need the following two lemmas, which show that some auxiliary relations can be maintained in first order. The first auxiliary relation we need is an almost total order on the nodes in an undirected graph.

LEMMA 4.1. *Let G be an undirected graph. We can maintain in first order a binary relation T on nodes in G and its transitive closure TC_T, where T satisfies the following conditions*:

1. *If G is empty then so is T.*

2. *If G contains exactly one edge, say $\langle a, b\rangle$, then T may be $\varnothing$, $\{(a, b)\}$, or $\{(b, a)\}$.*

3. *If $|G| > 1$, then either $TC_T(x, y)$ or $TC_T(y, x)$ but not both, whenever x and y are nodes in G such that $\langle x, y\rangle$ is not an isolated edge of G.*

Clearly, both T and TC_T are acyclic. The relation T we incrementally build is essentially the order of "arrival" with some "delayed decision," and the ordering will be used to choose an element from a set of elements, when necessary. An example of modifying T after insertions is given in Fig. 4.

We did not define T to be an exact successor relation on nodes of G, since it is unclear how such a relation could be maintained incrementally. Indeed, there are situations where we cannot make an ordering from the two newly inserted nodes. For example, when we insert an undirected edge $\langle a, b\rangle$ which is isolated in the new graph, there is no way to say in first order that a is before b or b is before a. When this happens we will only append both a and b to the tail end of the old T. (This is the reason why T is not a total order.) We delay the decision on the ordering of a and b

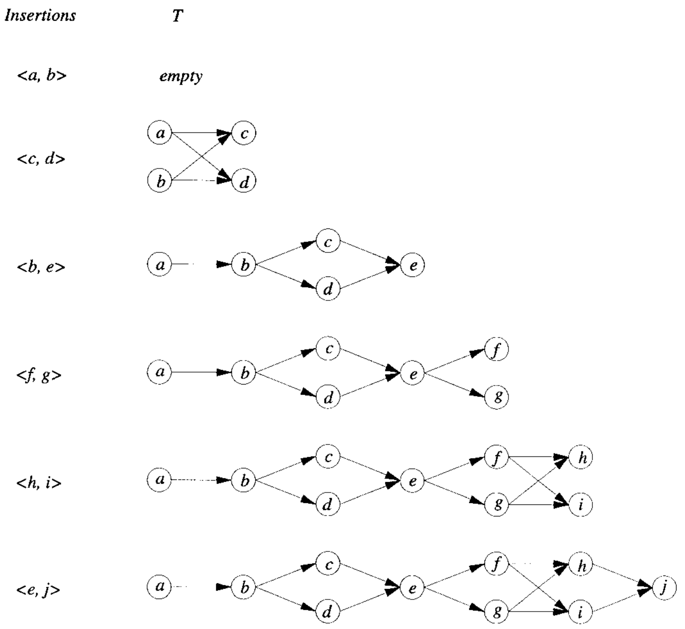


FIG. 4. An example of maintaining the "almost" successor relation T.

until this edge gets connected to other edges. However, when an edge becomes isolated after losing its connecting edges we will remember the "history" and keep their ordering.

Proof of Lemma 4.1. Since T is acyclic, TC_T can be maintained in first order as in Example 3.2, provided that T can be maintained in first order and each update to G only causes the insertion/deletion of a bounded number of tuples in T. As we will see below, we can indeed maintain T in such a way.

For the maintenance of T after insertions, suppose[2] we insert $\langle a, b\rangle$. If $\langle a, b\rangle$ is in G^o then we do nothing. Suppose $\langle a, b\rangle$ is not in G^o. Two cases arise:

Case 1. $T^o = \varnothing$. If $G^o = \varnothing$, then let $T = \varnothing$. If $|G^o| = 1$ and neither of a, b occurs in G^o, then let $T = \{(x, y), (z, y) \mid G^o(x, z),\ y \in \{a, b\}\}$. Otherwise, it must be the case that $|G^o| = 1$ and exactly one of a, b occurs in G^o. In this case, we can distinguish in first order all three nodes. Let e_1 be the edge in G^o. We define $T = \{(x, y), (y, z)\}$, where x is the node of G which occurs in e_1 and in $\langle a, b\rangle$, y is the other node of e_1, and z is the other node in $\langle a, b\rangle$.

Case 2. $T^o \neq \varnothing$. We need to consider the following three situations:

2a. Neither of a, b occurs in G^o. Then we add both a, b as the last elements to T: If y is the last element in T^o, we let $T = T^o \cup \{(y, a), (y, b)\}$; if there are two last elements of T^o, say y, z, we let $T = T^o \cup \{(u, v) \mid u \in \{y, z\}, v \in \{a, b\}\}$.

[2] Recall that we disallow self-loop edges of the form (x, x).

2b. Exactly one of a, b occurs in G^o. Without loss of generality, suppose a occurs in G^o and hence b is a new node. We first make b the sole last node in T. If a, c are *equivalent* in T^o, i.e.,

$$\forall x \quad (T^o(a, x) \Leftrightarrow T^o(c, x)) \wedge (T^o(x, a) \Leftrightarrow T^o(x, c)),$$

then $\langle a, c\rangle$ must be an isolated edge in G^o; when this happens, we break the equivalence of a, c, and make a the successor of c in T.

2c. Both a, b occur in G^o. To maintain T we may need to break one or two equivalences. If a, c are equivalent in T^o, then $c \neq b$ and we modify T^o by making a the successor of c in T. The case when b is involved in an equivalence relationship in T^o is dealt with similarly.

To maintain T after the deletion of an edge e, we simply remove from T^o those nodes in e that are not in any other edges and then make necessary "repairs" for these nodes (linking their predecessors with the successors). ▮

Using T and TC_T, we can maintain in first order the other auxiliary relations: a binary directed "generating" forest of an undirected graph and the transitive closure of the forest.

LEMMA 4.2. *Let G be an undirected graph. Then we can maintain in first order a directed forest H and its transitive closure TC_H, where $\{\langle u, v\rangle \mid (u, v) \in H\} \cup \{\langle u, v\rangle \mid \langle u, v\rangle$ is an isolated edge of G$\}$ is an undirected spanning forest of G. Furthermore, for each update to G, the corresponding update to H consists of* (a) *a possible deletion of an edge,* (b) *a possible change of directions of edges in a path, and* (c) *a possible insertion of a bounded number of edges.*

Proof. Suppose an edge $\langle a, b\rangle$ is inserted into G^o. The following three cases are possible.

Case A. Neither of a, b occurs in G^o (or equivalently, $\langle a, b\rangle$ is isolated in G). Then let $H = H^o$.

Case B. Exactly one of a, b occurs in G^o. There are two possibilities.

B1. Neither of a, b occurs in H^o. Then there exists exactly one edge e in G^o which is connected to $\langle a, b\rangle$ and is isolated in G^o. We choose the node that occurs in both $\langle a, b\rangle$ and e as the root of a new tree with these two edges, and add this tree to H^o to get H.

B2. Exactly one node in a, b occurs in H^o. Let a be the node that occurs in H^o; we set $H = H^o \cup \{(a, b)\}$.

Case C. Both a, b occur in G^o. The following three situations can happen.

C1. Neither of a, b occurs in H^o. By the assumption that the edge $\langle a, b\rangle$ is not in G^o, there must exist exactly two isolated edges $e = \langle a, c\rangle$ and $e' = \langle b, d\rangle$ in G^o. We choose the direction (a, b) of e such that $TC_{T^o}(a, b)$ is true, i.e., a precedes b in the order relation T^o. We then choose the (a, c) and (b, d) directions of e and e'. We construct a new tree with the directed edges (a, b), (a, c), (b, d) and add this tree to H^o to get the new forest H.

C2. Exactly one of a, b occurs in H^o. Let a be the node that occurs in H^o. Then there must be an isolated edge $\langle b, c\rangle$ in G^o. We insert (a, b) and (b, c) to H^o to get H.

C3. Both a, b occur in H^o. It is possible that a, b are in the same tree in H^o; in that case H^o is not modified. This can checked by examining TC_{H^o}: a, b are in the same tree iff they have a common ancestor. Suppose now that a, b are not in the same tree. Then the edge $\langle a, b\rangle$ connects two trees in H^o. We combined these two trees into a single new tree by the following three steps:

(a) Since a, b are already in H^o, $\langle a, b\rangle$ is not an isolated edge and a, b are not equivalent in T^o. We choose one direction of e, say (a, b), such that $T^o(a, b)$ holds.

(b) If b is not a root (a root has no incoming edges) in H^o, we rotate the tree containing b so that b becomes the root of the rotated tree. The rotation to make b the new root of the second tree consists of two parts, to modify H and TC_H, respectively. (Fig. 5 shows an example.)

- Modifying H. Suppose c is the root in the tree of H^o where b appears. The rotation is achieved by reversing the edges in H^o on the path between c and b; i.e., an edge $H^o(u, v)$ becomes $H(v, u)$ whenever $\widehat{TC}_{H^o}(c, u) \wedge \widehat{TC}_{H^o}(v, b)$. For the simple example in Fig. 5, (c, d), (d, b) in H^o become (d, c), (b, d) (respectively) in H. All other edges of H^o stay unchanged.

- Modifying TC_H. We next adjust TC_{H^o} to TC_H in response to the rotation of H by letting $T_b = (TC_{H^o} - Y) \cup Z$, where Y is the set of edges to be removed from TC_{H^o},

$$Y = \{(u, v) \in TC_{H^o} \mid \exists w(\widehat{TC}_{H^o}(c, u) \wedge TC_{H^o}(u, w) \wedge \widehat{TC}_{H^o}(w, b) \wedge \widehat{TC}_{T^o}(w, v))\},$$

and $Z = \{(u, v) \mid \varphi\}$ is the set of new edges added to TC_H and

$$\varphi = \widehat{TC}_{H^o}(c, v) \wedge TC_{H^o}(v, u) \wedge \widehat{TC}_{H^o}(u, b) \vee \exists w \begin{pmatrix} \widehat{TC}_{H^o}(c, w) \wedge TC_{H^o}(w, u) \wedge \\ \widehat{TC}_{H^o}(u, b) \wedge TC_{H^o}(w, v) \wedge \\ \neg(TC_{H^o}(v, u) \vee TC_{H^o}(u, v)) \end{pmatrix}.$$

All edges in $TC_{H^o} - Y$ stay unchanged. Fig. 6 illustrates the sets Y and Z. For Fig. 5, $Y = \{(c, d)\} \cup \{c, d\} \times \{b, e, f, g, h\}\}$ and Z contains (b, d), (b, c), and (d, c) by the first disjunct, and (b, d), (b, h), (b, i), (b, j), (d, i), and (d, j) by the second disjunct.

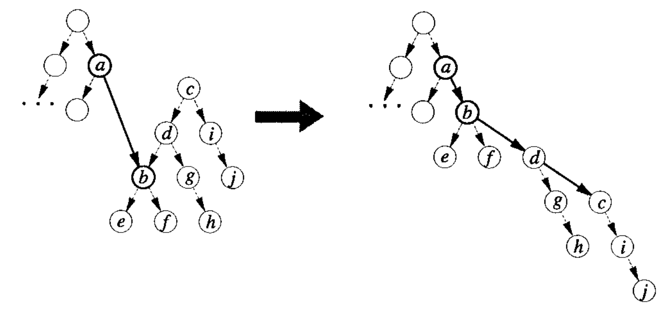


FIG. 5. An example of rotating a tree.

(c) Finally, we add the directed edge (a, b) to the forest produced by (b). Specifically, we add (a, b) to the forest (H) and derive TC_H from T_b and (a, b) using the formula provided in Example 3.1.

When an edge $\langle a, b\rangle$ is deleted, suppose one direction, say (a, b), of e is in H^o (otherwise we do nothing). Then this deletion breaks some tree in H^o into two. If there are edges in G connecting the two trees, we first choose one such edge $\langle c, d\rangle$ and choose a direction (c, d), both using the order provided in TC_T, and then insert (c, d) into the directed spanning forest H using the method described above. ▌

We are now ready to present the main results of the section concerning the connectivity and transitive closure queries over undirected graphs and the 2-colorability of undirected graphs. Here a graph G is said to be *2-colorable* if there is a partition of the notes into two sets X and Y such that each edge in G is incident to a node in X and a node in Y.

THEOREM 4.3. *The following queries belong to* FOIES$_2$ *but not to* FOIES$_1$.

1. *Connectivity and transitive closure over undirected graphs.*
2. *2-colorability of undirected graphs.*

Proof. We prove the membership of these queries in FOIES$_2$ here and leave the proofs of their nonmembership in FOIES$_1$ to Section 5.

Let G be the name of the undirected input graph. For both queries, we will use the following four common binary auxiliary relations (which are described in Lemmas 4.1 and 4.2): T, TC_T, H, TC_H. As mentioned earlier, TC_T is used to choose an element from a set of elements, when necessary. Their maintenance in first order was given in Lemmas 4.1 and 4.2.

Let $I = \{\langle u, v\rangle \mid \langle u, v\rangle$ is isolated in $G\}$ and $F = \{\langle u, v\rangle \mid (u, v) \in H\} \cup I$.

For (1), the connectivity and transitive closure queries over undirected graphs, we observe that $TC_G = TC_F$ and $TC_F(x, y) \equiv F(x, y) \vee \exists u(\widehat{TC}_H(u, x) \wedge \widehat{TC}_H(u, y))$. Connectivity is derivable from TC_G in first order.

For (2), the 2-colorability query, we define, for each binary relation R, a predicate $even_R$ as $even_R(x, y)$ iff there is an even-length walk from x to y in R. We define odd_R similarly. In addition to the four common binary relations, we also maintain $even_H$ and odd_H. Clearly, G is 2-colorable iff $\neg\exists xy(even_G(x, y) \wedge odd_G(x, y))$ holds. So it suffices to show that $even_H$ and odd_H can be maintained, and that $even_G$ and odd_G can be derived from $even_H$ and odd_H.

For maintaining $even_H$ and odd_H, we need to consider three cases: (i) H is obtained by inserting one edge into H^o; (ii) H is obtained by deleting one edge from H^o; (iii) H is obtained from H^o by changing directions of all edges in a path. (Corresponding to one update to G, we may need to compose several of these updates to $even_H$ and odd_H to reflect the complete change to the even/odd relations.)

Suppose (a, b) is inserted into H^o. Then, the new $even_H$ and odd_H can be expressed as

$$\begin{aligned} odd_H(x, y) \equiv{} & odd_{H^o}(x, y) \vee [\widehat{even}_{H^o}(x, a) \\ & \wedge \widehat{even}_{H^o}(b, y) \vee odd_{H^o}(x, a) \wedge odd_{H^o}(b, y)] \\ even_H(x, y) \equiv{} & even_{H^o}(x, y) \vee [odd_{H^o}(x, a) \wedge \widehat{even}_{H^o}(b, y) \\ & \vee \widehat{even}_{H^o}(x, a) \wedge odd_{H^o}(b, y)]. \end{aligned}$$

Suppose (a, b) is deleted from H^o. We compute new odd_H and $even_H$ by deleting al tuples (x, y) in odd_{H^o} and $even_{H^o}$ if there is a path from x to y through (a, b).

Suppose now that H is obtained from H^o by changing directions of all edges in a path. To fix the notation, let us say the path starts at c and terminates at b. We first delete

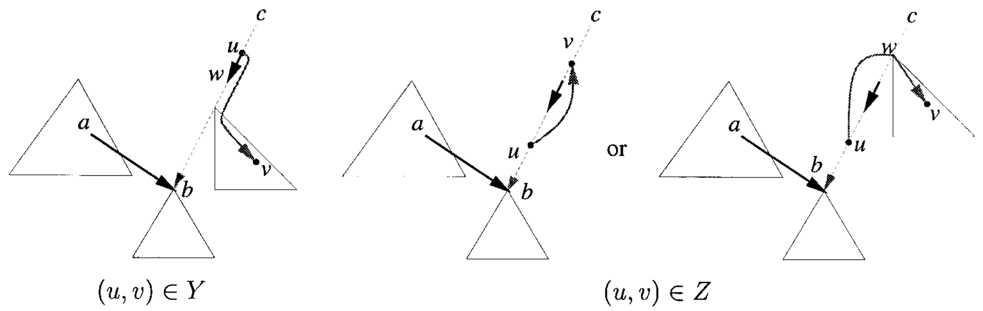


FIG. 6. Sets Y and Z used in modifying the transitive closure of H.

each pair (x, y) in the $even_{H^o}$ and odd_{H^o} relations where the only path in H^o from x to y includes a node in the path from c to b. (See Fig. 5.) Now we add pairs as follows:

- For all x, y both on the path from c to b, we simply reverse direction: insert $even_H(y, x)$ if $even_{H^o}(x, y)$ and insert $odd_H(y, x)$ if $odd_{H^o}(x, y)$.
- For all x on the path from c to b and all old descendants y of c such that the least common ancestor w of x and y is on the path from c to b, determine the parity of (x, y) by combining the parity of the path from w to x and the parity of the path from w to y using the four possible combinations.

We can express $even_F$ and odd_F as

$$\begin{aligned} even_F(x, y) \equiv{} & \widehat{even}_H(x, y) \vee \widehat{even}_H(y, x) \\ & \vee \exists u(even_H(u, w) \wedge even_H(u, y) \\ & \vee odd_H(u, x) \wedge odd_H(u, y)) \\ odd_F(x, y) \equiv{} & I(x, y) \vee odd_H(x, y) \vee odd_H(y, x) \\ & \vee \exists u(even_H(u, x) \wedge odd_H(u, y) \\ & \vee odd_H(u, x) \wedge even_H(u, y)). \end{aligned}$$

Then $even_G$ and odd_G can be expressed as

$$\begin{aligned} even_G(x, y) \equiv{} & even_F(x, y) \vee \exists uv(even_F(u, v) \wedge G(u, v) \\ & \wedge (\widehat{even}_F(x, u) \wedge odd_F(v, y) \\ & \vee odd_F(x, u) \wedge \widehat{even}_F(v, y))) \\ odd_G(x, y) \equiv{} & odd_F(x, y) \vee \exists uv(even_F(u, v) \wedge G(u, v) \\ & \wedge (\widehat{even}_F(x, u) \wedge \widehat{even}_F(v, y) \\ & \vee odd_F(x, u) \wedge odd_F(v, y))). \end{aligned}$$

We now verify that the expression for $even_G$ is correct. The proof for odd_G can be obtained from this proof by mostly interchanging the words "odd" and "even."

Observe that each of the three disjuncts in the right-hand side of the expression, namely $even_F(x, y)$, $\exists uv(even_F(u, v) \wedge G(u, v) \wedge \widehat{even}_F(x, u) \wedge odd_F(v, y))$, and $\exists uv(even_F(u, v) \wedge G(u, v) \wedge odd_F(x, u) \wedge \widehat{even}_F(v, y))$, exhibits a path of even length. So the right-hand side does not derive more than necessary. We now show that the right-hand side derives every pair of nodes between which there are even walks. Suppose x and y are two nodes and $u_0 u_1 \cdots u_n$ is a walk from x to y in G such that n is even. If $even_F(x, y)$ is true, then we are done. Suppose otherwise. Since F is an undirected spanning forest for G, $odd_F(x, y)$ must be true; so there exists a path from x to y of odd length. Due to the acyclicity of F, all walks in F from x to y are of odd length. Let $x_0 x_1 \cdots x_m$ be a walk in F from x to y which goes through all u_i's $(0 \leqslant i \leqslant n)$. (Such a walk exists since F is a spanning forest for G.) Hence m is odd and each $u_i = x_{j_i}$ for some j_i. Consider the following list of pairs: $(j_0, j_1), (j_1, j_2), ..., (j_{n-1}, j_n)$. There are an even number of them since n is even, but $j_0 = 0$ is even and $j_n = m$ is odd. It is easily seen that there must be at least one pair, say (j_k, j_{k+1}), in the list where j_k and j_{k+1} are both even or both odd. If they are both even, then $\exists uv(even_F(u, v) \wedge G(u, v) \wedge \widehat{even}_F(x, u) \wedge odd_F(v, y))$ is true; if they are both odd, then $\exists uv(even_F(u, v) \wedge G(u, v) \wedge odd_F(x, u) \wedge \widehat{even}_F(v, y))$ is true. In both cases the right-hand side holds for x and y. ∎

The foies construction for 2-colorability does not generalize for 3-colorability: A 3-colorable graph may have a pair of nodes between which there are three paths whose lengths modulo 3 are 0, 1, and 2. (Note that the existence of a foies for the 3-colorability query implies PTIME = NPTIME.)

The idea in the above proof can also be used to maintain minimum spanning trees for undirected graphs. (Over a ternary input relation, where the first two columns specify edges and the third column specifies the weight of the edge, the problem is to return a spanning forest with minimal weight.) The maintenance queries are constructed in a way similar to the ones used by Patnaik and Immerman [PI94] except that we derive their auxiliary P from our H and TC_H: $P(x, y, u)$ says u is on the unique path in F from x to y and is expressed as $\varphi(x, y, u) \vee \varphi(y, x, u)$, where $\varphi(x, y, u)$ is

$$TC_H(x, y) \wedge \widehat{TC}_H(x, u) \wedge \widehat{TC}_H(u, y) \bigvee \begin{pmatrix} \neg(\widehat{TC}_H(x, y) \vee \widehat{TC}_H(y, x)) \wedge \\ \exists v(\psi \wedge \widehat{TC}_H(v, x) \wedge \widehat{TC}_H(v, u) \wedge \widehat{TC}_H(u, y)) \end{pmatrix}$$

and ψ says that v is the least common ancestor of x and y.

THEOREM 4.4 1. *For* 1-*path directed graphs*, (s, t)-*connectivity and TC are in* FOIES_2 *but not in* FOIES_1.

2. *For acyclic directed graphs*, (s, t)-*connectivity*[3] *is in* FOIES_2 *but not in* FOIES_1, *while TC is in* $\mathrm{FOIES}_{\mathrm{sf}}$.

Proof. The negative results will be shown in Section 5. For (2), the positive results follow from Example 3.2. (That TC over acyclic graphs is in $\mathrm{FOIES}_{\mathrm{sf}}$ is proved in [DS95b]. Since by storing TC one can define (s, t)-connectivity, (s, t)-connectivity belongs to FOIES_2.) For (1), the 1-path case, we store two binary auxiliary relations: an arbitrary maximal acyclic subgraph H of G, and TC_H. Note that TC_H can be maintained as in Example 3.2 (and [DS95b]). We will use TC_H to help derive TC_G when an edge in a cycle of G is deleted.

Roughly, we maintain H as follow. When an edge is inserted into G^o, the edge is inserted into H^o iff this insertion

[3] For each graph G, the (s, t)-connectivity query returns *true* if there is a path from s to t in G.

does not introduce a cycle into H. When an edge e deleted from G^o is in H^o, we delete e from H^o; if there is another edge e' in $G^o - H^o$ such that e and e' are in a cycle of G^o (there is at most one such edge since G^o is 1-path), then we insert e' into H. Each time H is changed, the transitive closure of H is adjusted accordingly.

We maintain TC_G as follows. Insertions are handled as usual. Suppose $(a, b) \in G^o$ is deleted. If a and b are not in the same strongly connected component (SCC) of G^o, then we derive TC_G using a query in [DS95b] (an extension of the query in Example 3.2). This is similar to the case for acyclic graphs. Suppose a and b are in a common SCC of G^o (Fig. 7).

Let C be the set of all nodes in the same SCC of G^o as a, and let T_0 be TC_H limited to the nodes in C, i.e., $T_0 = \{(x, y) \in TC_H \mid x, y \in C\}$. The following steps will derive TC_G:

(i) Let T_1 be the result of removing from TC_{G^o} all pairs (x, y) such that there is a path from x to y through (a, b) in G^o, i.e., $T_1 = TC_{G^o} - \{(x, y) \mid \widehat{TC}_{G^o}(x, a) \wedge \widehat{TC}_{G^o}(b, y)\}$.

(ii) Then TC_G is defined by $\exists z_1 z_2[\hat{T}_1(x, z_1) \wedge \hat{T}_0(z_1, z_2) \wedge \hat{T}_1(z_2, y)]$.

For correctness, note that clearly we did not produce more pairs than necessary. To see that we produced everything we should, suppose x reaches y in G. The following three cases are possible.

(a) $\neg(\widehat{TC}_{G^o}(x, a) \wedge \widehat{TC}_{G^o}(b, y))$. Then $(x, y) \in T_1$.

(b) $\widehat{TC}_{G^o}(x, a) \wedge \widehat{TC}_{G^o}(b, y) \wedge \widehat{TC}_{G^o}(y, x)$. Then x and y are in the SCC of a in G^o, so $(x, y) \in T_0$.

(c) $\widehat{TC}_{G^o}(x, a) \wedge \widehat{TC}_{G^o}(b, y) \wedge \neg\widehat{TC}_{G^o}(y, x)$. Since G^o is 1-path, x can only reach y in G^o (and in G) through the SCC of G^o where a occurs. Let $u_0 u_1 \cdots u_m$ be the path from x to y in G. Clearly this is also a path in G^o. There must be some node in C which occurs in this path, since otherwise there would be two paths in G^o from x to y, one through some node in C and one not. It can be easily seen that all nodes in the path between two nodes in C must also be in C. This interval of nodes in C is covered by one reachability pair in H. If this interval consists of one node, we have either $T_1(x, y)$ true (this covers the case where either x or y is in the SCC of a in G^o) or $T_1(x, z) \wedge T_1(z, y)$ true; if this interval consists of mode nodes, then we have $\exists z[T_1(x, z) \wedge T_0(z, y) \vee T_0(x, z) \wedge T_1(z, y)] \vee \exists z_1, z_2[T_1(x, z_1) \wedge T_0(z_1, z_2) \wedge T_1(z_2, y)]$. ▮

For regular chain datalog queries, we can assert that they all have binary insertion-only foies [DT92, DST95], and

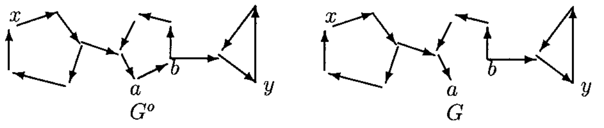


FIG. 7. 1-path edge deletion.

some of them (such as the transitive closure) do not have unary insertion-only foies. The negative part can be proved using the Ehrenfeucht–Fraïssé games developed in the next section over a long chain of edges roughly using the following argument: There must be two isomorphic segments with identical colors but first-order queries cannot tell the order of the two segments, yet the ordering is important since that corresponds to which segment can reach the other segment.

5. STRICTNESS INSIDE FOIES$_2$

In this section we prove results of the form "query Q is not in FOIES$_k$" for $k = 0$, 1, or "not in FOIES$_{\text{sf}}$." The proofs are based on Ehrenfeucht–Fraïssé games [Fra54, Ehr61]. Specifically, we show that *Mod-3* is not in FOIES$_{\text{sf}}$; *Equal-Length-Chain* is not in FOIES$_0$; and many other queries including 2-colorability, transitive closure over 1-path graphs and over undirected graphs, and various connectivity queries, etc., are not in FOIES$_1$. The game-based techniques presented here do not seem suitable for proving nonexistence of k-ary foies for $k \geqslant 2$. In the next section, we use a reduction to show that the FOIES$_k$ hierarchy is indeed strict.

This section is organized as follows. We first briefly describe the Ehrenfeucht–Fraïssé games and then use the games to prove nonexistence of k-ary foies for the above-mentioned queries for k being space free, 0,1 (in that order).

Ehrenfeucht–Fraïssé games are played on first-order structures. Let I and J be two (first-order) structures with universes U_I and U_J and r a positive integer. The *game of length r associated with I and J* is played by two players, a *spoiler* and a *duplicator*, each of which makes r moves. The spoiler starts by picking an element in U_I or U_J and the duplicator picks an element in the opposite structure. This is repeated r times. At each move, the spoiler has the choice of the structure, and the duplicator must respond in the opposite structure. Let a_i (b_i) be the ith element picked in U_I (U_J). The duplicator wins the round $\{(a_1, b_1), ..., (a_r, b_r)\}$ if the mapping $a_i \mapsto b_i$ is an isomorphism of the substructures $I/\{a_1, ..., a_r\}$ and $J/\{b_1, ..., b_r\}$. The duplicator *wins* the game of length r associated with I and J if the duplicator has a winning strategy; i.e., the duplicator can always win every game of length r on I and J, no matter how the spoiler plays. This is denoted by $I \equiv_r J$. Note that the relation $\equiv_r$ is an equivalence relation on structures.

Intuitively, $I \equiv_r J$ says that the structures I and J cannot be distinguished by looking at just r elements at a time. The main result concerning Ehrenfeucht–Fraïssé games states that the ability to distinguish among structures using games of length r is equivalent to the ability to distinguish among structures using first-order sentences of "quantifier depth" r (the maximum number of quantifiers in a path from the root to a leaf in the syntax tree of the sentence). In particular, Ehrenfeucht–Fraïssé's general result implies the following.

PROPOSITION 5.1. *Let φ be a first-order with quantifier depth r and I, J two structures. If $I \models \varphi$ and $I \equiv_r J$, then $J \models \varphi$.*

Ehrenfeucht–Fraïssé games and extensions are a very useful tool in proving the expressive power of query languages (e.g., [Fag75, AF90, AHV95, GS94]). But they cannot be used directly in proving results concerning foies, since the process of incremental evaluation is not strictly first order. Next, we extend Ehrenfeucht–Fraïssé games and prove that many queries discussed in the previous sections are outside classes FOIES_{sf}, FOIES_0, and FOIES_1.

Let I, J be two structures and f be a 1–1 function from a subset of U_I to a subset of U_J. We extend the Ehrenfeucht–Fraïssé games by adding the restriction that whenever the spoiler picks an element c such that either $f(c)=d$ or $f(d)=c$ for some d, the duplicator must pick d. We call the extended games *f-games* and denote by $I \equiv_r^f J$ that the duplicator has a winning strategy. Obviously, $I \equiv_r^f J$ implies $J \equiv_r^{f^{-1}} I$. If f is a function, we also denote by $\varphi[f]$ the formula obtained by replacing each occurrence of each constant c with $f(c)$. Proposition 5.1 can be extended to the following.

LEMMA 5.2. *Let φ be a sentence with quantifier depth r, I and J two structures, and f a 1–1 function from the set of constants in φ to U_J. If $I \models \varphi$ and $I \equiv_r^f J$, then $J \models \varphi[f]$.*

The proof of the above lemma is as follows. For the function f, we can construct unary relations, one for each constant, to encode f, i.e., to force the constants mapped by f to be matched against each other. The standard Ehrenfeucht–Fraïssé games can then be played and the result follows from Lemma 5.1.

We next present the proofs for the negative results stated earlier in Sections 3 and 4. We provide first the proof for FOIES_{sf} (*Mod-3*), then the proof for FOIES_0 (*Equal-Length-Chain*), and finally the proofs for FOIES_1 (2-colorability, *TC* over 1-path and over undirected graphs).

THEOREM 5.3. *The Mod-3 query is not in* FOIES_{sf}.

Proof. Suppose there exists a space-free foies $\mathscr{F}$ for *Mod-3* and let ϕ be the aux-maintaining query for the query answer in $\mathscr{F}$. Having only the answer to *Mod-3* available, $\mathscr{F}$ can only remember whether the current database size is a multiple of 3. Thus, when the size, say n, is not a multiple of 3, it cannot distinguish whether n satisfies $1 = n \bmod 3$ or $2 = n \bmod 3$. Below we use this fact to get a contradiction.

Let d be the quantifier depth of the first-order formula ϕ. Let R be the input unary relation to *Mod-3*, and I_1 and I_2 two instances of R such that $|I_i| \bmod 3 = i$ and $|I_i| > d$ for each $i = 1, 2$. Suppose that a is a new constant. Consider the insertion of a into I_i. Since *Mod-3*(I_i) is false for each $i = 1, 2$, we can replace *Mod-3*^{o} by *false*. We further replace each occurrence of the insertion set by a, i.e., "hardwire" a into the formula. Let the resulting formula be φ which now has the database (before insertion) as the only input. Since $\mathscr{F}$ is a foies for *Mod-3*, $\varphi(I_1)$ and $\varphi(I_2)$ should produce different answers. However, by construction $|I_i| > d$ for $i = 1, 2$, the duplicator has a winning strategy of d-round Ehrenfeucht–Fraïssé games on I_1 and I_2 [Kol95]. This implies that φ cannot distinguish I_1 and I_2, a clear contradiction. ∎

Space-free foies do not have auxiliary relations other than the query answer. The above proof simply exhibits two large databases such that the same update yields different answers but the first-order aux-maintaining query is incapable of telling the difference. We extend this idea to 0-ary and unary foies, i.e., foies with Boolean or unary auxiliary relations.

THEOREM 5.4. *The query* Equal-Length-Chain *is not in* FOIES_0.

Proof. Suppose $\mathscr{F} = (\mathscr{S}_{\text{aux}}, \alpha, \mathscr{A}^{\pm})$ is a 0-ary foies for *Equal-Length-Chain* where $\mathscr{S}_{\text{aux}}$ consists of m 0-ary relations which exclude the query answer (by definition), α is an aux-d-def, and $\mathscr{A}^{\pm}$ provides an aux-maintaining (first-order Boolean) query for each relation in $\mathscr{S}_{\text{aux}}$ and for the query answer. The idea of the proof is to find two databases I and I' and a sequence of insertions $\bar{\delta}$ such that $\bar{\delta}$ acting on I and I' yields different query answers but the first-order queries in $\mathscr{A}^{\pm}$ are incapable of distinguishing the difference.

Suppose d is the maximal quantifier depth of all first-order aux-maintaining queries in $\mathscr{A}^{\pm}$. Let $d' = d \cdot 2^m$. Since we shall consider a sequence of k cascading updates (for some fixed k), we "expand" the aux-maintaining queries in $\mathscr{A}^{\pm}$ to handle k updates at a time. We use $\mathscr{A}^{\pm k}$ to denote the expanded first-order queries.

For example, we can obtain the new aux-maintaining queries in $\mathscr{A}^{\pm 2}$ from the ones in $\mathscr{A}^{\pm}$ by replacing each occurrence of an auxiliary relation R by its aux-maintaining query $\mathscr{A}_R^{\pm}$ (which handles the first update). Then the queries in $\mathscr{A}^{\pm k}$ can be constructed by $k-1$ rounds of such replacements. It is easy to verify that the maximal quantifier depth of aux-maintaining queries for 2^m updates is $\leqslant d'$.

Recall that *Equal-Length-Chain* is defined over two binary relations q_1, q_2, each containing a chain. Now consider the following $2^m + 1$ databases, J_i, for $1 \leqslant i \leqslant 2^m + 1$, where

- J_1 consists of two chains q_1, q_2 of lengths ℓ, $\ell + 1$ (respectively) where $\ell \geqslant 2^{d'}$;
- for each $1 \leqslant i \leqslant 2^m$, J_{i+1} is obtained by inserting exactly one q_2-edge into J_i.

Clearly, since the length difference of the two chains in the database J_i is i, the answer *Equal-Length-Chain*(J_i) is false. Since $\mathscr{F}$ has only m additional Boolean auxiliary relations (excluding the query answer), there are at most 2^m different results for the auxiliary relations and there must exist two distinct databases $I, I' \in \{J_i \mid 1 \leqslant i \leqslant 2^m + 1\}$ such that $I(R) = I'(R)$ for each auxiliary relation $R \in \mathscr{S}_{\text{aux}}$; i.e., the auxiliary relations are identical for I and I'. Figure 8 illustrates a

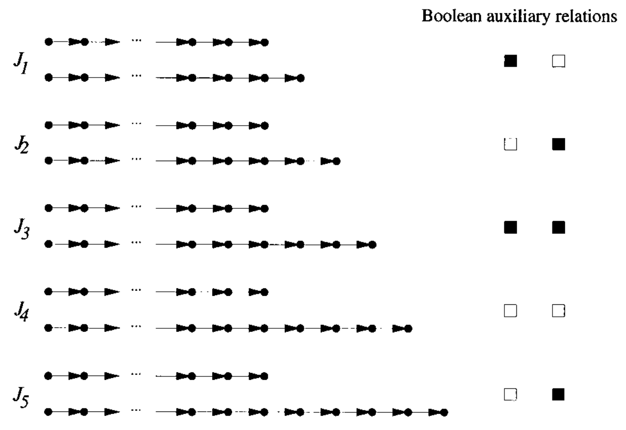


FIG. 8. The databases J_i when $m=2$.

possible case for $m=2$. Without loss of generality, assume $I=J_n$, $I'=J_{n'}$, and $n<n'$.

We now consider a sequence of n insertions $\bar{\delta}$ on $I$ and I' separately, each extending the q_1 chain by one edge. Since the length difference of the two chains in $I$ is n and in I' is $n'>n$, *Equal-Length-Chain*$(\bar{\delta}I)$ becomes *true*, while *Equal-Length-Chain*$(\bar{\delta}I')$ remains *false*. Suppose ϕ is the first-order aux-maintaining query maintaining the answer to *Equal-Length-Chain* for the case of n cascading updates. Because the quantifier depth of ϕ is $\leqslant d'$ and each chain in $I$, I' has length $\geqslant 2^{d'}$, it can be shown that the duplicator has a winning strategy when playing d'-round f-games on $I$ and I', where f is the identity mapping defined only on the nodes incident to the newly inserted edges. The winning strategy is similar to the winning strategy for the Ehrenfeucht–Fraïssé games on linear order [Kol95]. Hence $\bar{\delta}I \equiv_r^f \bar{\delta}I'$. This contradicts the fact that ϕ yields different results on $\bar{\delta}I$ and $\bar{\delta}I'$. ∎

We next consider the class FOIES_1. Recall that each unary foies may store a fixed number of unary auxiliary relations during the incremental evaluation. It is convenient to view these unary relations as colorings on the constants (nodes) in the input database. We first examine Boolean queries.

THEOREM 5.5. *The* 2-*colorability query is not in* FOIES_1.

Proof. Suppose that the Boolean query 2-colorability does have a unary foies $\mathscr{F}$, which uses m unary auxiliary relations. We further assume that d is the quantifier depth of the aux-maintaining query ϕ which updates the query answer.

Now consider the database $I$ consisting of n_1 simple chains, each of which has exactly n_2 nodes, where $n_2 \geqslant 4\cdot 3^d$ is an odd number and $n_1>(2^m)^{n_2}$. Since each unary auxiliary relation contains a set of nodes, we can view the combinations of the membership of a node in the m unary auxiliary relations as a "node color pattern." Having m unary auxiliary relations means that there are at most 2^m different node color patterns. Therefore for chains with n_2 nodes, there are at most $(2^m)^{n_2}$ chains with distinct color patterns. Because $I$ has $n_1>(2^m)^{n_2}$ number of chains of length n_2 there must be two chains C_1, C_2 which have the same color pattern (Fig. 9). More precisely, for each i, j, $1\leqslant i\leqslant n_2$ and $1\leqslant j\leqslant m$, the ith node of C_1 is in the jth auxiliary relation iff the ith node of C_2 is.

Let the first and last nodes of C_1 (C_2) be a_1, b_1 (respectively a_2, b_2). We now consider two separate insertions on $I$: one (δ_1) inserts the edge $\langle a_1, b_1\rangle$, while the other (δ_2) inserts the edge $\langle a_1, b_2\rangle$. Since $I$ is acyclic, $\delta_2 I$ is also acyclic and thus 2-colorable. But δ_1 introduces an odd cycle, so $\delta_1 I$ becomes non-2-colorable. However, since the difference occurs at the two chains having length exponential in the quantifier depth of the aux-maintaining query ϕ maintaining the query answers, it is impossible for ϕ to respond differently on the two insertions δ_1, δ_2. The argument is again based on d-round f-games where f maps the two nodes incident to the inserted edge in δ_1 to corresponding nodes incident to the inserted edge in δ_2. The duplicator's winning strategy is again quite similar to that used in the games for linear orders [Kol95]. ∎

THEOREM 5.6. *The transitive closure query over* 1-*path graphs is not in* FOIES_1.

Proof. The proof is accomplished by exhibiting a database (a 1-path graph) and two separate updates which cannot be distinguished by the first-order aux-maintaining queries of an assumed unary foies. The latter argument uses f-games.

Suppose the *TC* query over 1-path graphs is in FOIES_1. Let $\mathscr{F}=(\mathscr{S}_{\text{aux}}, \alpha, \mathscr{A}^{\pm})$ be a unary foies for this query, which has m unary auxiliary relations. Note that the query answer is a binary relation containing the transitive closure of the input database (graph). Let d be the quantifier depth of the first-order query ϕ in $\mathscr{A}^{\pm}$ which updates the query answer. In particular, $\phi(x, y)$ states that there is a path from x to y in the new graph.

We consider a database $I$ which is a huge cycle of at least $8n\cdot((2^m)^{2n}+1)$ nodes, where $n\geqslant 3^d$. We consider paths in $I$. Two paths of length $8n$ are called *n-similar* if their center segments of length $2n$ are isomorphic with respect to all stored unary (and of course Boolean) auxiliary relations (see Fig. 10). Clearly there are at most $(2^m)^{2n}$ non-n-similar paths of length $8n$ (analogous to the argument in the proof

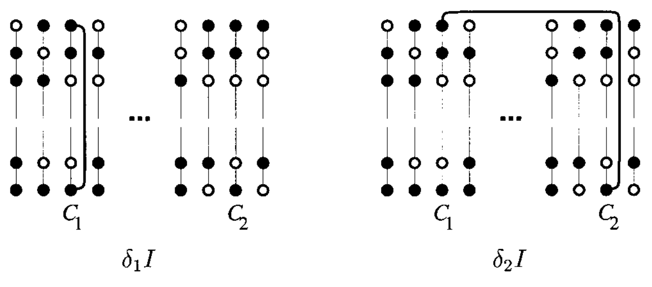


FIG. 9. The graph $I$ and updates δ_1, δ_2.

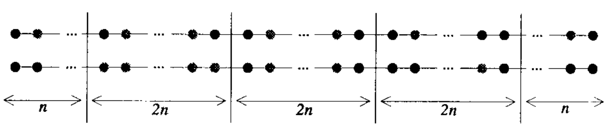


FIG. 10. Two similar paths of length $8n$.

of Theorem 5.5). Since I has at least $(2^m)^{2n}+1$ nonintersecting paths of length $8n$, there must exist two similar paths of length $8n$. Let J_1, J_2 be the two similar paths of length $8n$, (a_1, b_1) be the edge in (near) the center of J_1, and (a_2, b_2) be the edge in J_2 which is the isomorphic image of (a_1, b_1). Finally, let s, t be the center nodes of the paths from b_2 to a_1 and from b_1 to a_2 (respectively) (Fig. 11). Note that the distances between s and a_1, b_2 and between t and b_1, a_2 are at least $4n$.

Now consider the following two updates: δ_1 deletes the edge (a_1, b_1), δ_2 deletes the edge (a_2, b_2). It is obvious that there is a path from s to t in $\delta_2 I$ but not in $\delta_1 I$. This is the property we shall use in the following game-theoretic arguments.

The formula $\phi(x, y)$ contains several parameters: G, δ, $\bar{r}_0$, $\bar{r}_1$, and TC_G, where G is a database (graph), δ is an update, $\bar{r}_0$ and $\bar{r}_1$ are Boolean and unary auxiliary relations in $\mathscr{S}_{\text{aux}}$ with respect to G, and TC_G is the transitive closure of G. We modify the formula $\phi(s, t)$ by replacing the update δ by v_1, v_2 representing the deletion of (v_1, v_2), $\bar{r}_0$ by the appropriate values of *true* or *false* as given by the auxiliary relations, and $TC_G(u, w)$ by *true* for all (u, w). Suppose $\varphi_{G, \bar{r}_1}(s, t, v_1, v_2)$ is the resulting formula. The formula φ simply states that s remains connected to t in the new graph, G without the edge (v_1, v_2), if G is fully connected. We now argue that $\varphi_{I, \bar{r}_1}(s, t, a_1, b_1)$ and $\varphi_{I, \bar{r}_1}(s, t, a_2, b_2)$ can only be both *true* or both *false*. This clearly contradicts the fact that there is a path from s to t in $\delta_2 I$ but not $\delta_1 I$.

Let f be the mapping from $\{a_1, b_1\}$ to $\{a_2, b_2\}$ such that $f(a_1)=a_2$, $f(b_1)=b_2$. Then $\varphi_{I, \bar{r}_1}(s, t, a_2, b_2) = \varphi_{I, \bar{r}_1}(s, t, a_1, b_1)[f]$. Since φ has the same quantifier depth d as ϕ, we argue that $\delta_1 I \equiv_d^f \delta_2 I$, i.e., the duplicator has a winning strategy when playing d-round f-games on $\delta_1 I$ and $\delta_2 I$. The winning strategy of the duplicator is as follows. When the spoiler plays anywhere outside the center segments of J_1 or J_2, the duplicator plays by picking the same element in I; when the spoiler plays near a_1, b_1 (or a_2, b_2) the duplicator makes corresponding moves near a_2, b_2 (or a_1, b_1). Since the segment length $2n$ is exponential in the number of rounds to play, the spoiler will not tell the difference between the two segments nor make any connection from a_1, b_1, a_2, b_2 to s, t. The details of the strategy again resemble those for the connectivity query [Kol95]. ▮

Using the same approach with different graphs and update sequences, or reductions to queries known to be outside of FOIES_1, we can establish the following.

THEOREM 5.7. *The following queries are not in* FOIES_1: *the transitive closure of directed and undirected graphs*; *(s, t)-connectivity of* 1-*path and acyclic graphs*; *connectivity and disconnectivity of undirected graphs*; *every unbounded chain query*; *the same generation query*, *even over acyclic graphs*.

Proof. The result that transitive closure of directed graphs is not in FOIES_1 follows directly from Theorem 5.6. For transitive closure over undirected graphs, we use the graph (shown in Fig. 12) which consists of a large number of chains sharing a common end. Then there must exist two isomorphic chains with respect to all the auxiliary relations; suppose these two chains end at nodes s and t, respectively. Consider two different sequences of updates, one sequence consisting of inserting $\langle s, s' \rangle$ followed by deleting $\langle a, b \rangle$ and the other sequence consisting of inserting $\langle s, s' \rangle$ followed by deleting $\langle c, d \rangle$. It then can be argued using f-games that a foies cannot tell the difference between the two resulting graphs, while it is clear that one update sequence results in a connected graph and the other results in a disconnected graph. The results on connectivity and disconnectivity queries over

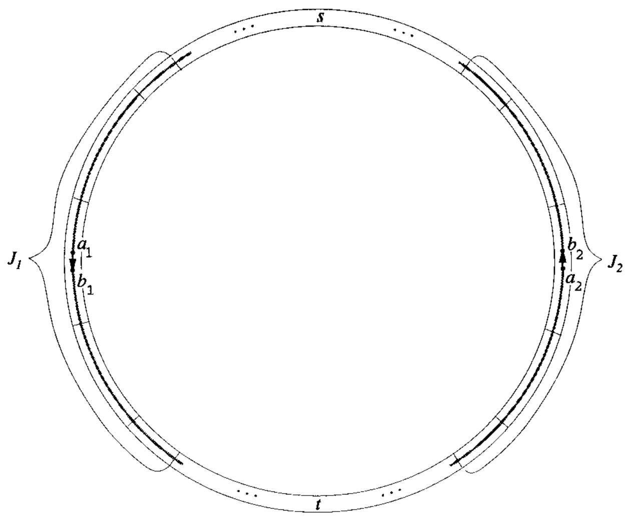


FIG. 11. The graph I used in the proof of Theorem 5.6.

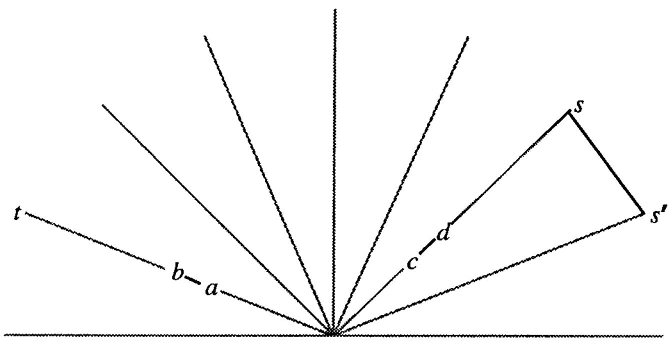


FIG. 12. An undirected graph.

various classes of graphs follow from the transitive closure query, except for the acyclic graphs.

For the (s, t)-connectivity of acyclic graphs, suppose $\mathscr{F}$ is a unary foies. We start with a long chain from s to t. Then there are two long intervals, far apart, say from a to b and from c to d, of the long chain which are isomorphic to each other with respect to all the auxiliary relations. We consider two update sequences: one inserts the edge (a, b) and then deletes the center edge of the interval from a to b, while the other inserts the edge (a, b) and then deletes the center edge of the interval from c to d. It is easy to verify that $\mathscr{F}$ cannot tell the difference between the two resulting graphs, while one resulting graph is (s, t)-connected and the other resulting graph is not. So $\mathscr{F}$ cannot be a foies for the (s, t)-connectivity of acyclic graphs.

The same generation query is defined on two binary relations (graphs) R_1, R_2 such that a pair (x, y) is in the answer if there exists a node u such that there are two paths of the same length: one from u to x in R_1 and the other from u to y in R_2. To show the same generation query is not in FOIES$_1$ we use the database shown in Fig. 13, where R_1 contains a chain of length n for some large n and R_2 contains a much longer chain of length m. Since the R_2 chain is very long, we can find two segments l_1, l_2 of length $n-1$ which are isomorphic to each other with respect to all the auxiliary relations. Upon the insertion of the edge e into R_2 (see Fig. 13), it can be established that either both (a, b), (a, c) are in the query answer or neither of them is in, a clear contradiction.

The above game-theoretic argument for the same generation query can be further extended to show that every unbounded chain query is not in FOIES$_1$. Since each chain query assumes that the database schema contains only binary relations, we can view the input database as an edge-labeled graph. Clearly, each chain query Q can be associated with a context-free grammar G_Q such that (x, y) is in the answer iff there is a path in the input graph from x to y with the labels (along the path) spell a word in the language of G_Q. Since Q is unbounded, the language of G_Q is infinite. We can then find a word in the language with length exponential in the maximal quantifier depth of first-order formulas in the assumed foies that maintain the query answer. The construction of the graph and updates follows an idea similar to that for the same generation query. ▮

Figure 14 summarizes results on some of the queries discussed in previous sections; results proven in this paper are italicized. Note that connectivity and transitive closure

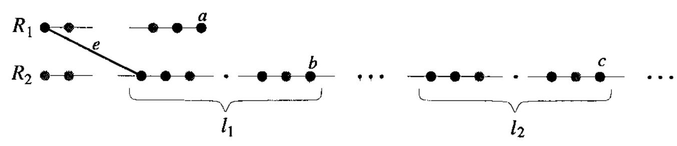
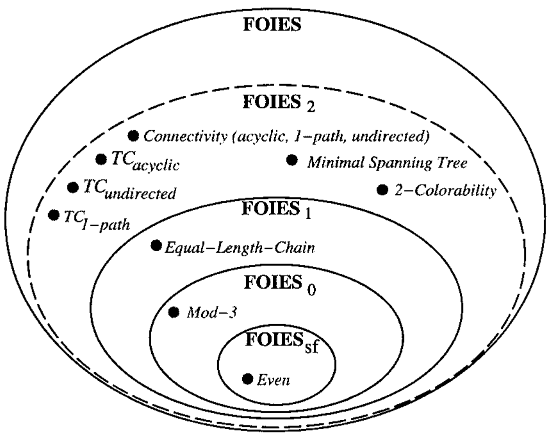


FIG. 13. A database for the same generation query.



Query	*Arity*	Space-Bounded FOIES				FOIES
		sf	0	1	2	
Even	0	Yes	Yes	Yes	Yes	Yes
Mod-3	0	*No*	*Yes*	Yes	Yes	Yes
Equal-Length-Chain	0	No	*No*	*Yes*	Yes	Yes
2-Colorability	0	No	No	*No*	*Yes*	Yes
TC	2	No	No	*No*	open	open
▷ acyclic	2	Yes	Yes	Yes	Yes	Yes
▷ 1-path	2	No	No	*No*	*Yes*	Yes
▷ undirected	2	No	No	*No*	*Yes*	Yes
(s,t)-connectivity	0	No	No	*No*	open	open
▷ acyclic	0	No	No	*No*	*Yes*	Yes
▷ 1-path	0	No	No	*No*	*Yes*	Yes
(Dis)connectivity: undirected	0	No	No	*No*	*Yes*	Yes

FIG. 14. Summary of queries and results.

are different (for undirected graphs) since the answers of the latter contain more information. However, it is easy to observe the following.

PROPOSITION 5.8. *If (s, t)-connectivity over a family of graphs belongs to* FOIES$_k$, *then the corresponding transitive closure query is in* FOIES$_{k+2}$.

As an aside, we note the following: The graphs and updates used in proving transitive closure over undirected graphs can also be used to show that there is no unary foies for on-line computation of minimal spanning trees of graphs. Indeed, when all edges have the same weight in the graph described above in Fig. 12, the two insertions above should lead to different results which the first-order formulas fail to tell.

6. THE FOIES$_k$ HIERARCHY IS STRICT

We now present a main result of this paper which states that the FOIES$_k$ hierarchy is indeed proper. The proof of this result also settles the open problem raised in our earlier paper [DS97] on the "deterministic" version of the FOIES$_k$

hierarchy. We also discuss the impact of the arity of the input databases on the arity hierarchies.

We show that $\text{FOIES}_k \subsetneq \text{FOIES}_{k+1}$ for each $k \in \mathbb{N}$. The proof uses a result by Cai [Cai90, Th. 3.2] which establishes an exponential lower bound on the size of constant-depth Boolean circuits with "help" bits to compute the "multiple parity" problem. We first state a weaker version of Cai's Theorem below.

THEOREM 6.1 [Cai90]. *Let $X = \{x_{ij} \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant m^5\}$ be a set of Boolean variables and C be an unbounded fan-in, constant depth circuit computing the m parity functions $x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{im^5}$ for $1 \leqslant i \leqslant m$. Suppose further that in addition to the X variable, the circuit C also takes as input $m-1$ arbitrary precomputed values, which can depend on the X variables in any desired way (these are called the* help bits). *Then the circuit C must have size $\geqslant 2^{m^\lambda}$, where λ is a constant depending on the depth of C.*

We now use the above theorem by Cai to prove that FOIES_k hierarchy is strict at each level. In particular, we encode the multiple parity problem into a query in FOIES_k and prove that the query cannot be in FOIES_{k-1}. The latter is shown by exhibiting a first-order solution to the multiple parity problem using some $(k-1)$-ary stored auxiliary relations, assuming the query is in FOIES_{k-1}. Since each first-order query (formula) can be computed by an unbounded fan-in, constant depth Boolean circuit of polynomial size, and auxiliary relations can be represented as help bits, this contradicts the exponential lower bound established in Theorem 6.1.

THEOREM 6.2. *For each integer $k > 0$, $\text{FOIES}_{k-1} \subsetneq \text{FOIES}_k$.*

Proof. Let $\mathscr{S}$ be the database schema with a single $(6k+1)$-ary relation R. We define the *k-Mod-4* query over $\mathscr{S}$ as follows: $\forall I \in inst(\mathscr{S})$, $k\text{-}Mod\text{-}4(I) = \{(x_1, ..., x_k) \mid N_{x_1, ..., x_k} \bmod 4 = 0\}$, where $N_{x_1, ..., x_k}$ is the cardinality of the set $\{(y_1, ..., y_{5k+1}) \mid R(x_1, ..., x_k, y_1, ..., y_{5k+1})\}$.

Observe that $k\text{-}Mod\text{-}4 \in \text{FOIES}_k$. Indeed, to maintain the answers to *k-Mod-4*, one only needs to keep three additional k-ary auxiliary relations r_i, for each $i \in \{1, 2, 3\}$, which keeps track of all $(x_1, ..., x_k)$ tuples in $\pi_{1, 2, ..., k}(R)$ such that $N_{x_1, ..., x_k} \bmod 4 = i$.

In the following, we prove that *k-Mod-4* cannot be in FOIES_{k-1}. Suppose otherwise, i.e., *k-Mod-4* has a $(k-1)$-ary foies $\mathscr{F}$. We first rewrite $\mathscr{F}$ into another $(k-1)$-ary foies $\mathscr{F}'$ that uses only $(k-1)$-ary auxiliary relations. Suppose that $\mathscr{F}'$ has μ auxiliary relations. We pick $n \in \mathbb{N}$ such that $n > \mu$. Suppose $D \subseteq \mathbf{dom}$ is a set of n constants and a_1, a_2, a_3, a_4 are four distinct elements in D.

Let P be a given instance of Cai's multiple parity problem with $m = n^k$. We will show how to solve the multiple parity problem using an AC^0 circuit, through the foies $\mathscr{F}'$ for *k-Mod-4*.

Since $m = n^k$, we identify each integer i $(1 \leqslant i \leqslant m)$ with a distinct k-ary tuple t_i in D^k and each integer j $(1 \leqslant j \leqslant m^5)$ with a distinct $5k$-ary tuples s_j in D^{5k}. Let I be the following database instance:

$$I(R) = (D^k \times \{a_1\}^{5k+1}) \cup \{(t_i, s_j, a_2), (t_i, s_j, a_3) \mid x_{i,j} \text{ is } \textit{true} \text{ in the instance } P\}.$$

Clearly $\pi_{1, ..., k} I(R) = D^k$. Observe that for each i $(1 \leqslant i \leqslant m)$, if $x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{im^5} = 1$ (or 0), then the number of tuples of form $(t_i, y_1, ..., y_{5k+1})$ is 3 mod 4 (or 1 mod 4, respectively). Thus $k\text{-}Mod\text{-}4(I) = \emptyset$.

For each i, $1 \leqslant i \leqslant m$, consider the update $\delta_i = (\triangle_i, \emptyset)$, where the insertion set $\triangle_i = \{(t_i, a_4, ..., a_4)\}$ and t_i is the tuple in D^k identified with i. Since $\mathscr{F}'$ is a foies for *k-Mod-4*, there is a first-order query $\phi_i(\delta, I, I_{\text{aux}})$ which computes the new answer (the current answer is empty). From the definition of the query *k-Mod-4* and the database I, it is clear that the new answer will be either the same as the current answer (when the number of tuples of form $(t_i, y_1, ..., y_{5k+1})$ is still not a multiple of 4) or the current answer plus the tuple t_i. By building the update δ_i into ϕ_i and using the construction of I, we can easily obtain a first-order formula $\varphi_i(I, I_{\text{aux}})$ which returns true iff $x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{im^5} = 0$.

We can now construct an AC^0 circuit C to compute the multiple parity problem with $m-1$ help bits. Let $r_1, ..., r_\mu$ be the μ auxiliary relations in $\mathscr{F}'$ for the database I. We can represent each relation r_i by n^{k-1} bits. Therefore, the auxiliary relations $r_1, ..., r_\mu$ can be represented by $\mu \cdot n^{k-1} \leqslant (m-1)$ bits, which will be the help bits. Now for each i, $1 \leqslant i \leqslant m$, we construct an AC^0 circuit C_i from φ_i to compute the parity of $x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{im^5}$. Since C is an AC^0 circuit, it has a polynomial size (in m), contradicting Theorem 6.1. ▌

In [DS97], we considered "deterministic" foies, which are foies with the restriction that the auxiliary relations are defined by (fixpoint) queries. This severely limits the power of foies. For example, a deterministic foies cannot maintain a total order of the active domain. In [DS97], we proved that the corresponding arity-based hierarchy det-FOIES_k is strict at small arities $\leqslant 2$ and conjectured that it is strict at each arity. The proof of the above theorem also shows that det-FOIES_k hierarchy is strict at each level, thus settling that conjecture.

THEOREM 6.3. *For each integer $k > 0$, det-$\text{FOIES}_{k-1} \subsetneq$ det-FOIES_k.*

It is interesting to note that the arity of the input database schema plays a role in the arity hierarchy. Let (det)-FOIES_k^ℓ denote the set of all queries over an ℓ-ary database schema that have k-ary (deterministic) foies. The following are refined statements of Theorems 6.2, 6.3, 3.5, and 4.3.

COROLLARY 6.4. 1. *For each integer* $k>0$, (det)-$\mathrm{FOIES}_{k-1}^{6k+1} \subsetneqq$(det)-$\mathrm{FOIES}_{k}^{6k+1}$.

2. $\mathrm{FOIES}_{\mathrm{sf}}^{1} \subsetneqq \mathrm{FOIES}_{0}^{1}$.
3. $\mathrm{FOIES}_{0}^{2} \subsetneqq \mathrm{FOIES}_{1}^{2}$.
4. $\mathrm{FOIES}_{1}^{2} \subsetneqq \mathrm{FOIES}_{2}^{2}$.

In [DZ97] the following improvement was given: (det)-$\mathrm{FOIES}_{k-1}^{3k+1} \subsetneqq$(det)-$\mathrm{FOIES}_{k}^{3k+1}$; this was achieved by tuning Cai's result and tuning the proof of Theorem 6.2.

In [DS97] it was shown that the det-FOIES_k hierarchy collapses at level 0 if the input arity is limited to 1: $\bigcup_k$ det-$\mathrm{FOIES}_{k}^{1} =$ det-FOIES_{0}^{1}. This result also shows that the deterministic and nondeterministic foies arity hierarchies behave differently if the input arity is limited to 1, since it can be easily shown that the *Count-kth-Power* query is in $\mathrm{FOIES}_{2}^{1} - \mathrm{FOIES}_{1}^{1}$ for all positive integers k. (The *Count-kth-Power* query takes as input two unary relations R and S and checks if $|R| = |S|^k$. Membership in FOIES_{2}^{1} can be proven in a way similar to that for Theorem 4.1 of [DW97].)

For the general input arity restricted situation, there are many interesting problems that remain unanswered, including the following:

- For each $k>0$ and $\ell>0$, is it the case that $\mathrm{FOIES}_{k}^{\ell} \subsetneqq \mathrm{FOIES}_{k+1}^{\ell}$? In particular, is the hierarchy strict for $\ell=2$, or even 1?
- For each $k>0$ and $\ell>0$, is it the case that det-$\mathrm{FOIES}_{k}^{\ell} \subsetneqq$det-$\mathrm{FOIES}_{k+1}^{\ell}$? In particular, is the hierarchy strict for $\ell=2$?

The same generation query over acyclic graphs and the transitive closure over k-path graphs might be useful in answering the above questions. They are in FOIES_{4}^{2} but not FOIES_{1}^{2}, proven in the Appendix, though it is open whether they belong to FOIES_{2}^{2} or FOIES_{3}^{2}.

PROPOSITION 6.5. *The following two queries belong to* FOIES_{4}^{2}: *the same generation query over acyclic graphs and the transitive closure query over k-path graphs.*

7. FIRST-ORDER INCREMENTAL DEFINABILITY

In the current definition of a foies, storing the query answer is "free," regardless of its size (arity). An alternative way to measure the space usage of foies is to charge not only for storing auxiliary relations but also for storing the answer to the query of interest. This allows us to concentrate on the space cost of the incremental "definition" of a query.

One may be tempted in saying that a query Q can be incremental defined using arity cost n if Q as a k-ary foies, Q has arity m, and $n = \max\{k, m\}$. There are two considerations against this naive definition. (i) The measure under this definition does not reflect the difficulty of maintaining the answer of the query. For example, first-order queries, even those with the same conceptual difficulty, are not equally easy to maintain: letting Q_{id}^{i} be the (identity) query defined by $Q_{\mathrm{id}}^{i}(r) = r$ for each i-ary relation r, then Q_{id}^{k} could be incrementally defined using arity k but Q_{id}^{k+1} could not. (ii) The measure under this definition does not properly reflect the difficulty of maintaining the non-first-order part of the query, since it may be distorted by the first-order part of the query. Indeed, consider the k-ary ($k>0$) query Q_0 defined by $Q_0(r) = r$ if the number of tuples in r is even and $Q_0(r) = \varnothing$ otherwise. Under the naive measure, the arity cost of the incremental definition of Q_0 would be k. However, Q_0 can be defined as the composition of two queries $Q_1 Q_2$, where Q_2 tests if r is even, and Q_1 will produce r (or the empty set) if Q_2 sets (respectively, does not set) a flag. If we maintain the non-first-order part Q_2 of Q_0 only, then we can derive the answer to Q_0 as well. So the real incremental definition cost for Q_0 should be 0 instead of k.

The following definition addresses these two issues. Intuitively, the auxiliary database is incrementally maintained as in foies; however, after the maintenance for an update, a first-order query can be applied to the input database and the auxiliary database to derive the answer to the query Q.

DEFINITION 7.1. Let Q be a query from $\mathscr{S}_{\mathrm{in}}$ to R_Q and $k \in \mathbb{N}$. A *k-ary first-order incremental definition* (*foid*) for Q is a quadruple $(\mathscr{S}_{\mathrm{aux}}, \alpha, \mathscr{A}^{\pm}, \psi)$ satisfying the following conditions:

1. $\mathscr{S}_{\mathrm{aux}}$ is a k-ary database schema disjoint from $\mathscr{S}_{\mathrm{in}} \cup \{R_Q\}$;
2. α is an input domain preserving aux-d-def from $\mathscr{S}_{\mathrm{in}}$ to $\mathscr{S}_{\mathrm{aux}}$;
3. $\mathscr{A}^{\pm}$ maps each relation name R in $\mathscr{S}_{\mathrm{aux}}$ to a first-order aux-maintaining query $\mathscr{A}_{R}^{\pm}$;
4. for each database I_{in}^{o} in $dom(Q)$, each permissible update $\delta = (\triangle, \triangledown)$, and each reachable auxiliary database I_{aux}^{o} in $\alpha(I_{\mathrm{in}}^{o})$, if $I_{\mathrm{in}}^{n} = \delta I_{\mathrm{in}}^{o}$ is the new database, then a new auxiliary database I_{aux}^{n} in $\alpha(I_{\mathrm{in}}^{n})$ is obtained by

$$I_{\mathrm{aux}}^{n}(R) = \mathscr{A}_{R}^{\pm}(I_{\mathrm{in}}^{o}, I_{\mathrm{aux}}^{o}, \triangle, \triangledown) \qquad \text{for each} \quad R \in \mathscr{S}_{\mathrm{aux}};$$

5. ψ is a first-order query from $\mathscr{S}_{\mathrm{in}} \cup \mathscr{S}_{\mathrm{aux}}$ to R_Q such that $\psi(I, J) = Q(I)$ for each $I \in dom(Q)$ and each $J \in \alpha(I)$.

Similarly, a foid is called *space free* if $\mathscr{S}_{\mathrm{aux}} = \varnothing$.

Remark. A foid for query Q differs from a foies for Q in the following three aspects:

1. The answer to Q must be stored in the foies but is not necessarily stored in the foid.
2. The arity of the query answer is not counted toward the arity of the foies, but is counted toward the arity of the foid if the foid stores the answer.
3. Since the query answer may or may not be stored in the foid, the answer is computed by applying a first-order query on the current database and stored relations (if any).

The arity-based FOID_k hierarchy is defined in a way similar to the FOIES_k hierarchy.

Definition 7.2. For each $k \in \mathbb{N}$, let FOID_k be the set of all queries which have k-ary foids. Let FOID_{sf} be the set of queries which have space-free foids.

Observe that FOID_{sf} is precisely the set of all first-order queries. So the identity queries are in FOID_{sf}. Moreover, the query Q_0 considered prior to the definition of a foid is in FOID_0. The FOID_k classes also enjoy the following closure properties similar to Proposition 2.5.

Proposition 7.3. *For each $k \in \mathbb{N}$, FOID_k is closed under the first-order operations: projection, selection, cross product, set operations, and complement.*

Proof. Let $\mathscr{F}_1$, $\mathscr{F}_2$ be two k-ary foids for queries Q_1, Q_2 (respectively). Consider a binary operation θ. We can easily combine $\mathscr{F}_1$, $\mathscr{F}_2$ into a foid for $Q_1 \theta Q_2$ by retaining all aux-maintaining queries in $\mathscr{F}_1$, $\mathscr{F}_2$ and constructing easily the aux-maintaining query for $Q_1 \theta Q_2$. Unary operations are similar. ∎

The FOID_k classes are different from the FOIES_k classes. For example, *Even* is in FOIES_{sf} (Example 2.3) but not in FOID_{sf} since the latter coincides with first-order queries. The following can be easily verified.

Theorem 7.4. 1. $\text{FOID}_{\text{sf}} \subsetneq \text{FOIES}_{\text{sf}}$.

2. *For each $k \in \mathbb{N}$, $\text{FOID}_k \subseteq \text{FOIES}_k$; furthermore, the containment is proper for $k \leqslant 1$.*

3. *If an m-ary query Q is in FOIES_k then Q is in FOID_n for some $n \leqslant \max\{k, m\}$. It follows that a Boolean query is in FOIES_k iff it is in FOID_k.*

The strictness in (1) and (2) of Theorem 7.4 can be established using the transitive closure query over acyclic graphs. To verify (3), it suffices to observe that we can add the answer R_Q to Q as an auxiliary relation and use the identity first-order query on R_Q as the ψ.

Although the FOIES_k and FOID_k hierarchies do not coincide, they only differ slightly. Almost all results on FOIES_k reported in Sections 3, 4, and 5 can be translated into ones on FOID_k. In particular, the following hold.

Theorem 7.5. 1. *For all $k \in \mathbb{N}$, $\text{FOID}_k \subsetneq \text{FOID}_{k+1}$.*

2. *Mod-3 and Even are in FOID_0.*

3. *Equal-Length-Chain is in $\text{FOID}_1 - \text{FOID}_0$.*

4. *2-Colorability, (s, t)-connectivity for both acyclic graphs and 1-path graphs, (dis)connectivity for undirected graphs, and TC of acyclic (or undirected or 1-path) graphs are in $\text{FOID}_2 - \text{FOID}_1$.*

The proof for (1) above is almost identical to that of Theorem 6.2, using the *k-Mod-4* query as a separator. Items (2) and (3) follow from Theorem 7.4.(3). The membership part of (4) is clear; the proofs for the nonmembership part can be modified from the respective proofs of Section 5.

8. CONCLUDING REMARKS AND OPEN PROBLEMS

Our study on the arity of auxiliary relations brings out an interesting perspective on the incremental evaluation and definition of fixpoint queries using first-order queries. The results presented here clarify (i) the strictness of the evaluation arity hierarchy (FOIES_k) when the arities are small and when the input relations can be as large as $6k+1$, where k is the maximum arity of the auxiliary relations of the corresponding foies, and (ii) the strictness of the definition arity hierarchy (FOID_k). It should be pointed out that for the small arities the hierarchies are separated using graph queries, whereas for large arities the hierarchies are separated using queries over relations with large arities.

We also have the exact positions of several graph queries in the hierarchies, including some counting queries and the transitive closure of undirected graphs, acyclic graphs, and 1-path graphs.

There are many more open problems. We now list five:

• As mentioned earlier, recent improvement by [DZ97] states that we can separate FOIES_{k-1} and FOIES_k using queries over $(3k+1)$-ary input databases. It is still open whether we can separate FOIES_{k-1} and FOIES_k using queries over $f(k)$-ary input databases, where f is some function such that $f(k) \leqslant 3k$, $f(k) \leqslant k$, or $f(k) \leqslant 2$.

• Is the transitive closure query over directed graphs in FOIES?

• Is there a polynomial time complete problem in FOIES? (Please refer to the discussion on Dyn-FO toward the end of the Introduction for reasons that the existence of a PTIME-complete problem in Dyn-FO does not immediately imply the same for FOIES.)

• Does the same generation query have an (insertion-only) foies for arbitrary input relations? Is the same generation query for acyclic graphs in FOIES_2?

APPENDIX

In this Appendix we prove Proposition 6.5, which is restated in the following two propositions.

Proposition 1. *The same generation query over acyclic graphs belongs to FOIES_4^2.*

Proof. Suppose the two input relations are q_1 and q_2. We use $sl(x, y, u, v)$ to mean that there is a q_1-path from x to y that has the same length of a q_2-path from u to v. The sg relation can be derived as $sg(x, y) \equiv \exists z \; sl(z, x, z, y)$.

Upon the insertion of a new edge into q_1 (the case for q_2 is symmetric and thus omitted), the new sl relation is the union of its old value sl^o and the set Y, where $(x, y, u, v) \in Y$ iff

$$\begin{aligned} & q_1^{\triangle}(x, y) \wedge q_2^o(u, v) \\ & \quad \vee \exists u'(sl^o(x, w, u, u') \wedge q_1^{\triangle}(w, y) \wedge q_2^o(u', v)) \\ & \quad \vee \exists v'(q_1^{\triangle}(x, z) \wedge q_2^o(u, v') \wedge sl^o(z, y, v', v)) \\ & \quad \vee \exists u'v'(sl^o(x, w, u, u') \wedge q_1^{\triangle}(w, z) \\ & \qquad \wedge q_2^o(u', v') \wedge sl^o(z, y, v', v)), \end{aligned}$$

where $q_1^{\triangle}$ is the insertion set (containing a single edge).

We now consider how to maintain sl after the deletion of an edge (u_1, u_2) from q_1. Let T_1 denote the result of deleting all tuples (x, y, u, v) from sl^o where

$$\begin{aligned} & \exists u_1 u_2(\exists wz\, sl^o(x, u_1, w, z) \\ & \qquad \wedge q_1^o(u_1, u_2) \wedge \exists wz\, sl^o(u_2, y, w, z)) \end{aligned}$$

is true. Then we get the new sl in the following manner: $(x, y, u, v) \in sl$ iff

$$\begin{aligned} & T_1(x, y, u, v) \\ & \quad \vee \exists zw(q_1(x, z) \wedge q_2(u, w) \wedge T_1(z, y, w, v)) \\ & \quad \vee \exists zw(T_1(x, z, u, w) \wedge q_1(z, y) \wedge q_2(w, v)) \\ & \quad \vee \exists z_1 z_2 w_1 w_2(T_1(x, z_1, u, w_1) \wedge q_1(z_1, z_2) \\ & \qquad \wedge q_2(w_1, w_2) \wedge T_1(z_2, y, w_2, v)). \end{aligned}$$

To verify that this is correct, clearly the right-hand side does not derive more tuples than necessary. To see that it derives every tuple required, suppose x, y, u, $v$ are four nodes such that there exists a q_1-path $v_0 v_1 \cdots v_k$ from x to y and a q_2-path $w_0 w_1 \cdots w_k$ from u to $v$ with equal length. If there is no q_1^o-path going through (u_1, u_2), then $T_1(x, y, u, v)$ is true, and the proof is complete. Suppose there is a q_1^o-path from x to y through (u_1, u_2). It can be shown that (see [DP97] for a proof) there exists an $i < k$ such that there are no walks in q_1^o using (u_1, u_2) from v_0 to v_i or from v_{i+1} to v_k. The cases when $i = 0$ or $i = k-1$ are two degenerate cases of the case of $0 < i < k-1$, so we consider the case when $0 < i < k-1$. Clearly, $T_1(x, v_i, u, w_1)$, $T_1(v_{i+1}, y, w_2, v)$, $q_1(v_i, v_{i+1})$, $q_2(w_i, w_{i+1})$ all hold. Therefore $sl(x, y, u, v)$ can be obtained using the right-hand side. ▮

PROPOSITION 2. *The transitive closure query over k-path graphs is in* FOIES_4^2.

Proof. We will maintain k 4-ary relations, which store up to k distinct paths for each pair of nodes. In particular, we use the formula $sp_i(x, y, u, v)$ to mean that the edge (u, v) is on the ith path from x to y. For example, the ith path from a_0 to a_n

$$(a_0, a_1), ..., (a_{n-1}, a_n)$$

is stored as the following n tuples in sp_i:

$$\begin{matrix} (a_0 & a_n & a_0 & a_1) \\ & & \cdots & \\ (a_0 & a_n & a_j & a_{j+1}) \\ & & \cdots & \\ (a_0 & a_n & a_{n-1} & a_n) \end{matrix}$$

If an edge (c_1, c_2) is deleted, we simply delete (for each i) all tuples (x, y, u, v) in sp_i^o such that (x, y, c_1, c_2) is also in sp_i^o. Note that there is no need to split the ith path (from x to y). To help maintain the sp_i's after insertions we also do some compacting so that a consecutive initial segments of the identifiers is used.

Suppose a new edge (c_1, c_2) is inserted. Let x and y be two arbitrary nodes in the graph. If there are a path (with identified) m from x to c_1 and a path n from c_2 to y, we concatenate the two paths and the new edge to form a walk from x to y. The walk is a new path from x to y iff no cycle exists in the walk (Fig. 15), or equivalently iff path m and path n do not share nodes. (If a cycle exists in the walk then a path from x to y must already exist which consists of a subset of edges in the walk, e.g., the path from x to y via o and o' in Fig. 15.) This walk contributes to a new path iff the condition specified by the following first-order formula is true[4]:

$$\begin{aligned} & \neg \exists u((sp_m^o(x, c_1, u, *) \vee sp_m^o(x, c_1, *, u)) \\ & \qquad \wedge (sp_n^o(c_2, y, u, *) \vee sp_n^o(c_2, y, *, u))). \end{aligned}$$

We will use $\varphi_{mn}(x, y)$ to denote the formula saying: there are a path m from x to c_1 and a path n from c_2 to y which do not share any node.

We then use a systematic way to derive the sp_i relations. This can be done by first defining $sp_i(x, y, v, w)$ as $sp_i^o(x, y, v, w)$ whenever such an old path exists. After such copying, we use an order on pairs (m, n), where $\varphi_{mn}(x, y)$ holds, to "merge" the old contents of paths m and n and the edge (c_1, c_2) to define sp_i for the next i, i.e., sp_i is defined to contain

$$\begin{aligned} & (sp_m^o(x, c_1, v, w) \vee (v = c_1 \wedge w = c_2) \\ & \qquad \vee sp_n^o(c_2, y, v, w)) \wedge \varphi_{mn}(x, y). \end{aligned}$$

[4] We will use $sp_m^o(x, y, u, *)$ as a shorthand for the formula $\exists v\, sp_m^o(x, y, u, v)$, and $sp_m^o(x, y, *, *)$ for $\exists uv\, sp_m^o(x, y, u, v)$.

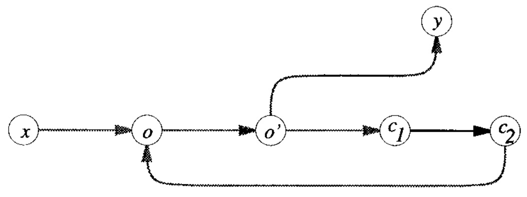


FIG. 15. A cycle introduced by the new edge.

The orderings of the steps can be coded into the formulas, although this is tedious. ▮

ACKNOWLEDGMENTS

The authors are grateful to Victor Vianu for interesting discussions and to Serge Abiteboul for his comments. We also thank the referees for their valuable comments. Most importantly, we are deeply indebted to Peter Bro Miltersen, who provided us with the proof of the strictness of FOIES$_k$ (Theorem 6.2) using Cai's theorem and pointed out the need of considering charging for the cost of storing the answer to the query of interest.

REFERENCES

[AF90] M. Ajtai and R. Fagin, Reachability is harder for directed than for undirected finite graphs, *J. Symbolic Logic* **55** (1990), 113–150.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu, "Foundations of Databases," Addison–Wesley, Reading, MA, 1995.

[AP87] K. R. Apt and J.-M. Pugin, Maintenance of stratified databases viewed as a belief revision system, *in* "Proceedings, ACM Symposium on Principles of Databases systems," pp. 136–145, 1987.

[BC93] A. L. Buchsbaum and M. C. Carlisle, Determining uniconnectivity in directed graphs, *Inform. Process. Lett.* **48** (1993), 9–12.

[BLT86] J. A. Blakeley, P.-A. Larson, and F. W. Tompa, Efficiency updating materialized views, *in* "Proceedings, ACM SIGMOD International Conference on Management of Data," pp. 61–71, 1986.

[Cai90] J. Cai, Lower bounds for constant-depth circuits in the presence of help bits, *Inform. Process. Lett.* **36** (1990), 79–83.

[CH82] A. K. Chandra and D. Harel, Structure and complexity of relational queries, *J. Comput. System Sci.* **25** (1982), 99–128.

[DK97] G. Dong and R. Kotagiri, Incrementally evaluating constrained transitive closure by conjunctive querie, *in* "International Conference on Deductive and Object-Oriented Databases," 1997.

[DLW95] G. Dong, L. Libkin, and L. Wong, On impossibility of decremental recomputation of recursive queries in relational calculus and SQL, *in* "Proceedings, International Workshop on Database Programming Languages," 1997.

[DP97] G. Dong and C. Pang, Maintaining transitive closure in first-order after node-set and edge-set deletions, *Inform. Process. Lett.* **62** (1997), 193–199.

[DS93] G. Dong and J. Su, First-order incremental evaluation of datalog queries (extended abstract), *in* "Proceedings, 4th International Workshop on Database Programming Languages," 1993.

[DS95a] G. Dong and J. Su, Increment boundedness and nonrecursive incremental evaluation of datalog queries (extended abstract), *in* "Database Theory—ICDT'95" (G. Gottlob and M. Y. Vardi, Eds.), Lecture Notes in Computer Science, Vol. 893, pp. 397–410, Springer-Verlag, Berlin/New York, 1995.

[DS95b] G. Dong and J. Su, Incremental and decremental evaluation of transitive closure by first-order queries, *Inform. Comput.* **120** (1995), 101–106.

[DS97] G. Dong and J. Su, Deterministic foies are strictly weaker, *Ann. Math. Artificial Intelligence* **19** (1997).

[DST95] G. Dong, J. Su, and R. Topor, Nonrecursive incremental evaluation of datalog queries, *Ann. Math. Artificial Intelligence* **14** (1995), 187–223; Also appeared as Uni Melbourne CS TR 93/3, March 1993.

[DT92] G. Dong and R. Topor, Incremental evaluation of datalog queries, *in* "Proceedings, International Conference on Database Theory," pp. 282–296, Berlin, Germany, 1992.

[DW97] G. Dong and L. Wong, Some relationships between foies and $\sum_1^1$ arity hierarchies, *Bull. European Assoc. Theoret. Comput. Sci.* **61** (1997).

[DZ97] G. Dong and L. Zhang, Separating auxiliary arity hierarchy of first-order incremental evaluation using $(3+1)$-ary input relations, Technical report, 97/13, 8 pages, Computer Science Department, University of Melbourne, 1997.

[Ehr61] A. Ehrenfeucht, An application of games to the completeness problem for formalized theories, *Fund. Math.* **49** (1961).

[Fag75] R. Fagin, Monadic generalized spectra, *Z. Math. Logik Grund. Math.* **21** (1975), 89–96.

[Fra54] R. Fraïssé, Sur les classifications des systèmes de relations, *Publ. Sci. Univ. Alger, I* **1** (1954).

[FSV95] R. Fagin, L. J. Stockmeyer, and M. Y. Vardi, On monadic NP vs monadic co-NP, *Inform. Comput.* **120** (1995), 78–93.

[GL95] T. Griffin and L. Libkin, Incremental maintenance of views with duplicates, *in* "Proceedings, ACM SIGMOD International Conference on Management of Data," 1995.

[GMS93] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, Maintaining views incrementally, *in* "Proceedings, ACM SIGMOD International Conference on Management of Data," pp. 157–166, 1993.

[GS94] S. Grumbach and J. Su, Finitely representable databases, *in* "Proceedings, ACM Symposium on Principles of Database Systems," 1994.

[Hul86] R. Hull, Relative information capacity of simple relational schemata, *SIAM J. Comput.* **15** (1986), 856–886.

[Kol95] P. G. Kolaitis, A tutorial on combinatorial games in database theory, *in* "Proceedings, ACM Symposium on Principles of Database Systems," 1995.

[Küc91] V. Küchenhoff, On the efficient computation of the difference between consecutive database states, *in* "Proceedings, Second International Conference on Deductive Object-Oriented Databases" (C. Delobel, M. Kifer, and Y. Masunaga, Eds.), Lecture Notes in Computer Science, Vol. 566, pp. 478–502, Springer-Verlag, Berlin/New York, 1991.

[MSVT94] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia, Complexity models for incremental computation, *Theor. Comput. Sci.* **130** (1994), 203–236.

[PI94] S. Patnaik and N. Immerman, Dyn-FO: A parallel dynamic complexity class, *in* "Proceedings, ACM Symposium on Principles of Database Systems," pp. 210–221, 1994.

[RRSS94] R. Ramakrishnan, K. A. Ross, D. Srivastava, and S. Sudarshan, Efficient incremental evaluation of queries with aggregation, *in* "International Logic Programming Symposium," 1994.

[WDSY91] O. Wolfson, H. M. Dewan, S. J. Stolfo, and Y. Yemini, Incremental evaluation of rules and its relationship to parallelism, *in* "Proceedings, ACM SIGMOD International Conference on Management of Data," pp. 78–87, 1991.