Deterministic FOIES are strictly weaker

Guozhu Dong∗

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

Jianwen Su∗∗

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

After a bounded update to a database, a first-order incremental evaluation system (abbreviated foies) derives the new answer to an expensive database query by applying a first-order query on the old answer and perhaps some stored auxiliary relations. The auxiliary relations are also maintained in first order. A foies can be deterministic or nondeterministic, depending on whether its (stored) auxiliary relations are defined by deterministic or nondeterministic mappings (from databases). In this paper we study the impact of the determinism restriction on foies and we compare nondeterminism with determinism in foies. It turns out that nondeterministic foies are more powerful than the deterministic ones: deterministic foies using auxiliary relations with arity ≤ k are shown to be strictly weaker than their nondeterministic counterparts for each k ≥ 1, and it is shown that there is a simple query which has a nondeterministic foies with binary auxiliary relations but does not have any deterministic foies with auxiliary relations of any arity. A strict arity hierarchy of deterministic foies is established for the small arities (≤ 2). Interestingly, the deterministic foies arity hierarchy collapses to 0-ary when limited to queries over unary relations.

1. Introduction

Over the past decade, there has been wide interest in database query languages and in query evaluation and optimization strategies, against databases that are “static” or not changing. As database systems extend their roles in data-intensive applications (e.g., active, scientific, spatial databases) [5], it is often the case that a vast amount of information need be processed with a relatively insufficient amount of computation resources and results/decisions be provided in real time. In spite of the technology development, drastic improvement of computation power enough to meet the needs is

∗ This author gratefully acknowledges support of the Australian Research Council through research grants. Part work done while on study leave at UCSB.
∗∗ Work by this author supported in part by NSF grants IRI-9109520 and IRI-9411330 and NASA grant NAGW-3888. Part work done while visiting the University of Melbourne.

© J.C. Baltzer AG, Science Publishers
hardly in the foreseeable future. It is increasingly urgent that new and more efficient mechanisms/strategies for database operations be developed and applied.

On the other hand, databases evolve: the information contained in databases changes over time, and each database instance is very often the accumulative result of a sequence of small updates rather than the result of a “big bang” from nothing. Despite many success stories in dealing with static databases, there is a serious lack of knowledge about the “dynamic” aspect and of necessary technologies to handle evolving databases effectively and efficiently. Recently, incremental evaluation of database queries (or the maintenance of views) has received considerable attention [2, 6–15, 19, 20, 22–26]. The fundamental problem here is to propagate, in a timely fashion, updates properly to the affected derived part of the database which are stored for efficiency reasons.

A framework of evaluating recursive queries (maintaining recursive views) by incrementally evaluating some non recursive queries (e.g., relational calculus queries), called “foies” or “Dyn-FO”, has recently been developed and studied [10–16, 23]. In this approach, auxiliary relations may be computed and stored (in addition to the answer to the query of interest); when an update occurs, the answer to the query and the new auxiliary relations on the new database are obtained using nonrecursive queries on the new database, the old auxiliary relations and the old answer. Using this approach, recursive views are maintained in the relational system using relational queries, although they are not directly computable using relational queries [3]. This approach also has great potential for parallel evaluations because relational queries have constant parallel complexity [1] and are thus readily adaptable to parallel implementations. Such an approach is useful for situations where the database changes frequently and the query answer must be computed in a real time manner. It has been shown that many database queries, including regular chain datalog with a particular kind of initializations [14] and generalized forms [11], transitive closure over acyclic graphs [12] and over undirected graphs [13, 23], can be evaluated in this way.

In [13] we use the maximal arity of the auxiliary relations in a foies as a measure to study the space efficiency of 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 (active domain of the) input database. Thus the maximal arity of auxiliary relations is an indicator of how hard it is to maintain the query using foies. A strict hierarchy was established for arities no larger than 2. The proofs for the positive results are constructive, while that for the negative results use Ehrenfeucht–Fraîssé games. Due to the combinatorial complexity of the Ehrenfeucht–Fraîssé game techniques it is still open whether the hierarchy is strict from arity 2 onwards.

In general, foies are “nondeterministic” since their (stored) auxiliary relations are defined by nondeterministic mappings from databases to relations. If the auxiliary relations are required to be defined by deterministic mappings, then such a foies is called “deterministic”. In this paper, we study the impact of this determinism restriction and compare deterministic foies with nondeterministic ones.

On the implementation side, deterministic foies are “scalable” in the sense that from a given deterministic foies which handles small updates (e.g., insertions and