A More Aggressive Use Of Views To Extract Information [*]

Nam Huyn[†]
Stanford University
huyn@cs.stanford.edu

Abstract

Much recent work has focussed on using views to evaluate queries. More specifically, queries are rewritten to refer to views instead of the base relations over which the queries were originally written. The motivation is that the views represent the only ways in which some information source may be accessed. Another use of views that has been overlooked becomes important especially when no equivalent rewriting of a query in terms of views is possible: even though we cannot use the views to get all the answers to the query, we can still use them to deduce as many answers as possible. In many global information applications, the notion of equivalence used is often too restrictive. We propose a notion of pseudo-equivalence that allows more queries to be rewritten usefully: we show that if a query has an equivalent rewriting, the query also has a pseudo-equivalent rewriting. The converse is not true in general. In particular, when the views are conjunctive, we show that all Datalog queries over the source do have a pseudo-equivalent Datalog query over the views. We reduce the problem of finding pseudo-equivalent queries to that of rewriting Horn queries with Skolem functions as Datalog queries. We present an algorithm for the class of term-bounded Horn queries. We discuss extending the problem to larger classes of Horn queries, other non-Horn queries that result from "inverting" Datalog views and adding functional dependencies. The theory and methods developed in our work have important uses in query mediation between heterogeneous sources, automatic join discovery and view updates.

[*]This work was written in October 1995 but has never been published before taking the current form of a technical report. The history can be traced as follows. This work was derived from [Hu96], a paper I wrote in October 1994 and whose original ideas were shared with Xiaolei Qian earlier in 1994. Qian's [Qian96] embodies some of these main ideas. The ideas from [Qian96] were in turn picked up by Oliver Duschka and used in [DG96] to derive what is essentially equivalent to our result.

[†]Work supported by ARO grant DAAH04–95–1–0192.

$$\begin{tikzcd}
\mathcal{V} \arrow[r, "Q'"] & \text{“same” answers} \\
\mathcal{R} \arrow[u, "\mathcal{M}"] \arrow[ur, "Q"'] &
\end{tikzcd}$$

Figure 1: Answering Queries Using Views

1 Introduction and Motivation

The problem of using views to answer queries has received a lot of attention recently ([LMSS95, CKPS95, RSU95]). A common framework used in these works as well as in this paper is depicted in Figure 1. In this figure, $\mathcal{R}$ represents a collection of base relations, $\mathcal{V}$ represents a collection of views, and $\mathcal{M}$ is a collection of view definitions that define the views in terms of the base relations. Given a query Q posed to $\mathcal{R}$, the problem is to find a query Q' that produces the “same” results as Q but that uses only the views in $\mathcal{V}$.

The approaches used in [LMSS95, CKPS95, RSU95] essentially look for “exact matches” Q' for Q: the answers produced by Q' from $\mathcal{V}$ must be exactly those answers Q would have produced from $\mathcal{R}$, had we had access to $\mathcal{R}$. The motivation is that $\mathcal{V}$ represents the only ways in which $\mathcal{R}$ may be accessed.

While the importance of using views to find exact matches cannot be overemphasized, another use of views that has been overlooked becomes crucial especially when no equivalent rewriting of a query in terms of views is possible: even though we cannot use the views to get all the answers to the query, we can still use them to infer as many answers as possible.

EXAMPLE 1.1 (Simple graph search) Consider a database $\mathcal{R} = (P_1, P_2)$ (with the corresponding predicates p_1 and p_2), a collection $\mathcal{V}$ consisting of a single view V (with predicate v) defined on $\mathcal{R}$ by the rule:

$$v(X,Y) \quad :- \quad p_1(X,Z),\ p_2(Z,Y).$$

and a query Q (with predicate q):

$$q(X) \quad :- \quad p_1(X,Y),\ p_2(Z,X).$$

Imagine a directed graph whose edges labelled p_1 (resp. p_2) denote the relation P_1 (resp. P_2). V represents patterns in a graph where the head of a p_1-edge coincides with the tail of a p_2-edge, while Q searches for patterns where the tail of a p_1-edge coincides with the head of a p_2-edge (see the two leftmost graphs in Figure 2).

Clearly, V alone cannot provide an exact match for Q. To see this, the rightmost graph in Figure 2 shows an instance for $\mathcal{R}$ where V becomes empty but $Q = \{c\}$. However, one can still use V to deduce answers for Q. For instance if we know $v(a,c)$ and $v(c,e)$, we can deduce $q(c)$, and this is the best we can do for Q. In a sense, in the absence of any exact match, the query Q' (with predicate p') defined by

$$q'(X) \quad :- \quad v(Y,X),\ v(X,Z).$$

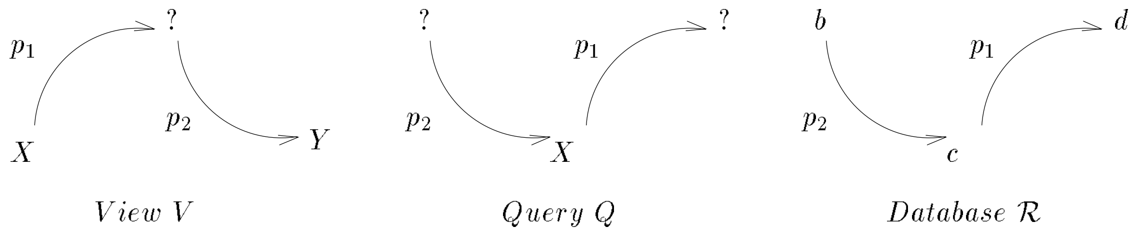

Figure 2: Searching a graph

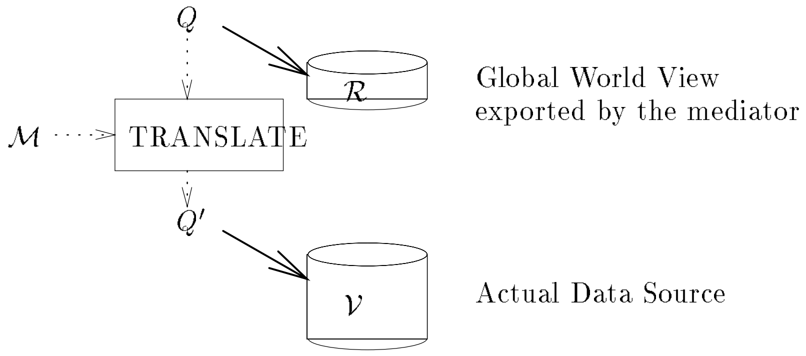

Figure 3: Query Mediation

can still serve as a good substitute for Q. It turns out that Q' is actually the "best" match for Q. ∎

This more aggressive use of views to extract information is further motivated by situations in which $\mathcal{V}$ represents the real information sources and $\mathcal{R}$'s role is to merely provide the predicates that give a global meaning to the predicates in $\mathcal{V}$ and to any global query Q. Such situations are found when trying to integrate heterogeneous sources [Chaw94, Qian93, Wie92].

EXAMPLE 1.2 (Integrating heterogeneous sources) To its end users, a mediator ([Chaw94, Wie92]) gives the illusion of a homogeneous source of information while drawing on actual data sources that are autonomous in many ways: schema design contents, participation, and query processing capability.

Figure 3 shows a mediator designed to export a certain global view of the world $\mathcal{R}$, usually within a narrowly scoped domain. A query Q, posed to the mediator in terms of $\mathcal{R}$, must be translated into a query Q' that the actual data sources $\mathcal{V}$ can process. In order to carry out the translation, the mediator has access to mappings $\mathcal{M}$ that define the source schemas (in $\mathcal{V}$) in terms of the global schema (in $\mathcal{R}$). Data sources may be individually mapped, and a new mapping is created when a new data source is "mounted" the first time to the mediator.

In order to be useful, mediators must maximize the information returned as answers to queries. The goal is to return all answers that can be deduced based on the contents of the sources and their query processing capability, all available mappings and natural constraints in the global view. ∎

Examples 1.1 and 1.2 suggest a new notion of query *pseudo-equivalence* that is clearly not the same as the notion used in the other approaches searching for an exact match.

Definition 1.1 (Pseudo-Equivalence):

Let $\mathcal{V}$ and $\mathcal{R}$ be two sets of relations, and let $\mathcal{M}$ be a set of view definitions that express the relations in $\mathcal{V}$ in terms of the relations in $\mathcal{R}$. Let Q be a query over $\mathcal{R}$ and Q' a query over $\mathcal{V}$. We say Q' is pseudo-equivalent to Q if

$$\forall \mathcal{V} : Q'(\mathcal{V}) = \bigcap_{\mathcal{R} \ s.t. \ \mathcal{V}=\mathcal{M}(\mathcal{R})} Q(\mathcal{R})$$

▮

Query Q' produces all the results that are logically entailed by $\mathcal{V}$, the view definitions in M that constrain R, and the query Q. Note that in this formulation of pseudo-equivalence, the model that varies is $\mathcal{V}$, while $\mathcal{R}$ is constrained, *though not uniquely in general*, by $\mathcal{V}$. By contrast, in the traditional notion of equivalence, defined by $[\forall \mathcal{R} : Q'(\mathcal{M}(\mathcal{R})) = Q(\mathcal{R})]$, $\mathcal{R}$ varies while $\mathcal{V}$ is *uniquely* defined by $\mathcal{R}$. But how are the two notions related?

Proposition 1.1 *If Q has an equivalent query Q', then Q also has a pseudo-equivalent query (in fact, Q' is pseudo-equivalent to Q). The converse is not true.* ▮

Proof: (Sketch) Assume that there is a Q' equivalent to Q. That is, for all $\mathcal{R}$, $Q'(\mathcal{M}(\mathcal{R})) = Q(\mathcal{R})$. It is sufficient to show that Q' is also pseudo-equivalent to Q. Let us fix $\mathcal{V}$ and let $\mathcal{R}$ be a database such that $\mathcal{V} = \mathcal{M}(\mathcal{R})$. Then $Q'(\mathcal{V}) = Q'(\mathcal{M}(\mathcal{R}))$. By hypothesis, Q' is equivalent to Q and thus, $Q'(\mathcal{M}(\mathcal{R})) = Q(\mathcal{R})$. So $Q'(\mathcal{V}) = Q(\mathcal{R})$ for any $\mathcal{R}$ such that $\mathcal{V} = \mathcal{M}(\mathcal{R})$.

To the converse, we use Example 1.1 as a counterexample. We already mentioned why no query over $\mathcal{V}$ equivalent to Q could exist. For the rest, it suffices to show that Q', as defined in Example 1.1, is pseudo-equivalent to Q. Assume $X \in Q'(\mathcal{V})$. Then any database $\mathcal{R}$ that satisfies $\mathcal{V} = \mathcal{M}(\mathcal{R})$ must contain tuples of the form $p_1(?1, ?2)$, $p_2(?2, X)$, $p_1(X, ?3)$, $p_2(?3, ?4)$, and therefore tuple $q(X)$ must be in Q and $X \in Q(\mathcal{R})$. Conversely, assume $X \in Q(\mathcal{R})$ for any $\mathcal{R}$ satisfying $\mathcal{V} = \mathcal{M}(\mathcal{R})$. Any such $\mathcal{R}$ must contain tuples of the form $p_2(?1, X)$, $p_1(X, ?2)$. We claim that $\mathcal{V}$ must contain tuples of the form $v(X, ?)$ since otherwise, it is consistent to have some $\mathcal{R}$ that does not contain any tuple of the form $p_1(X, ?)$ while still satisfying $\mathcal{V} = \mathcal{M}(\mathcal{R})$, contradicting what we just said. Similarly, $\mathcal{V}$ must also contains tuples of the form $v(?, X)$. Thus, $q'(X)$ must be in Q' and $X \in Q'(\mathcal{V})$. ▮

Challenges

The problem of finding pseudo-equivalent queries has three parameters – the class $\mathcal{L}_{\mathcal{M}}$ of mappings between $\mathcal{R}$ and $\mathcal{V}$, the class $\mathcal{L}_Q$ of input queries Q, and the class $\mathcal{L}_{Q'}$ of pseudo-equivalent queries Q' – that together, bear on the complexity of the problem. While $\mathcal{L}_{\mathcal{M}}$, $\mathcal{L}_Q$ and $\mathcal{L}_{Q'}$ are usually dictated by the application's requirements, it is important to develop the theory that would steer us away from the undecidable cases (see Theorem 6.1). We will therefore generally restrict ourselves to various subclasses of Datalog.

Results

In this paper, we establish the following:

- Under "conjunctive" mappings $\mathcal{M}$, any nonrecursive Datalog query Q has a pseudo-equivalent Q' in nonrecursive Datalog. We give a closed-form solution for Q'.
- For mappings $\mathcal{M}$ whose "inverses" $\mathcal{M}^{-1}$ are recursive but "term-bounded" Horn programs, we give an algorithm that essentially rewrite any query Q in recursive Datalog to a pseudo-equivalent Q' in recursive Datalog. Special cases include $\mathcal{M}$'s that are conjunctive and $\mathcal{M}^{-1}$'s that are nonrecursive Horn programs.

Related work

One approach to query translation between semantically heterogeneous data sources consists of extending the data model to match the expressive power of the query language enriched to capture the more complex mappings. DeMichiel's work [DeM89] on extended relational model and operations exemplifies this approach where indefinite answers to queries are allowed.

In spirit, Qian's query mediation approach to semantic interoperation [Qian93] is related to ours, in which source and target queries are expressed in a traditional first order query language and are required to return definite answers. However, translatability issues and computational aspects of query reformulation are not addressed in her work.

Incomplete deductive databases have been studied in [Im86] and [KL88] but with an emphasis on query evaluation. In [KL88], Kifer et al. identified a class of Horn queries that can be efficiently evaluated which traditional top-down or bottom-up evaluation techniques would fail to terminate. In [Im86], Imielinski studied the complexity of processing different classes of Horn queries and presented both positive and negative results. While negative results in query evaluability can be clearly carried over to our translation problem, it is much less clear how positive results would apply here.

Applications

The theory and methods developed here for finding pseudo-equivalent queries would not only provide a powerful approach to extract information from heterogeneous sources and to fully achieve their meaningful integration, but also have other interesting uses:

EXAMPLE 1.3 (Automatic join discovery) Consider a book mediator who exports the global schema *bookworld(Isbn,Title,Publisher)*. The mediator is given access to two real data sources: a reference source with relation *ref(Isbn,Title)*, and a locator source with relation *publish(Isbn,Company)*. The alert reader would immediate note that relation *bookworld* is nothing but the join of *ref* and *publish*. But in reality, having to prespecify all potential joins is neither desirable nor practical, especially when the underlying data sources can be independently "rolled" in. In fact, all we need to give the following definitions to the mediator:

$$\begin{aligned} ref(I,T) &\ \text{:-}\ \ bookworld(I,T,P). \\ publish(I,C) &\ \text{:-}\ \ bookworld(I,T,C). \end{aligned}$$

The mediator should be able to exploit the natural constraints that apply to the world of books, namely the functional dependencies

$$\begin{aligned} bookworld.Isbn &\rightarrow bookworld.Title \\ bookworld.Isbn &\rightarrow bookworld.Publisher \end{aligned}$$

to infer the following join formulation of *bookworld*

$$bookworld(I,T,P) \quad \text{:-} \quad ref(I,T),\ publish(I,P).$$

∎

EXAMPLE 1.4 (Approach to view updates) Consider a database $\mathcal{R}$ and a set of views $\mathcal{V}$ over $\mathcal{R}$, both of which allow updates. Assume that some of the views in V are not an updatable view, say, inserting tuples into some view would not result in unambiguously updating $\mathcal{R}$. One way to handle the view update problem is to extend the data model for $\mathcal{R}$. A better alternative is to leave the inserted tuples in $\mathcal{V}$ but not propagate them to $\mathcal{R}$. Use of these inserted tuples is delayed until query evaluation time. If we have techniques to translate queries against $\mathcal{R}$ into pseudo-equivalent queries against $\mathcal{V}$, we know that these inserted tuples will be fully considered in computing the query results. If $\mathcal{V}$ is fully maintained, we do not even need to consult $\mathcal{R}$ in order to find complete answers to queries. ∎

EXAMPLE 1.5 (Query simplification) As we will show later, our query translation technique essentially consists of precomputing all the potential unifications between subgoals and rule heads all of whose forms are known independently of the actual EDB extension. The structure of the query thusly translated has considerably been simplified and enables the potential of traditional query optimization techniques to be fully realized. ∎

Paper Outline

The rest of this paper is organized as follows. In the next section, we provide the basics for Datalog, mappings and terminology conventions used. In Section 3, we reduce the problem of finding queries Q' (pseudo-equivalent to Q) to that of computing the function-free answers of Horn programs Q^H. We also present a closed-form solution for the special case where the views in $\mathcal{V}$ and Q are conjunctive queries. In Section 4, we consider the general case where Q^H are recursive but "term-bounded" Horn programs and present an algorithm that rewrites Q^H to Q'. Finally in Section 6, we summarize our work and discuss possible extensions.

2 Preliminaries

Datalog and Horn programs

In this paper, queries and mappings are expressed by Datalog (and possibly Horn) programs. A Datalog program is a collection of Horn rules that have a single positive atom in the head and positive atoms (called subgoals) in the body. The only difference between Datalog programs and Horn programs is that function symbols are not allowed in the former but are allowed in the latter. The rules are assumed to be safe (see [Ull88]). Predicates used in a Datalog program are partitioned into EDB predicates and IDB predicates. An EDB predicate is a predicate that does not appear in rule's head (that is, only occurs in rule bodies). An IDB predicate is a predicate that appears in some rule's head (and possibly in rule bodies). Among the IDB predicates, there is a distinguished predicate called the *query predicate* that generates answers for the query. A conjunctive query is a Datalog program with only one rule. A query in nonrecursive Datalog can always be expanded into (and thus modelled by) a union of conjunctive queries.

Skolemization

In first order logic, when an existentially quantified variable is under the scope of some set of variables, it is legitimate to replace all occurrences of the former variable by a term, and this replacement process is call skolemization. The term's function is a new function symbol known as a Skolem function, and the remaining subterms are all the variables defining the scope. For instance, the sentence $(\forall X, Y)(\exists Z)p(X, Y, Z)$ is rewritten as $(\forall X, Y)p(X, Y, f(X, Y))$ after skolemization, using a new Skolem function symbol f. Skolemization is used to eliminated existential quantifications without loss of information.

Mappings

Generally, a mapping is a collection of sentences that define $\mathcal{V}$'s predicates in terms of $\mathcal{R}$'s predicates, and also capture semantics constraints in $\mathcal{R}$. For the purpose of this paper, the sentences are Datalog programs.

Queries

We use the standard notion of queries whose answers are ground atoms. In other words, we do not consider queries with disjunctive answers or answers containing quantified variables.

3 Using inverse mappings to find pseudo-equivalents

Inverse Mappings

Given a collection $\mathcal{V}$ of relations and a collection $\mathcal{M}$ of view definitions that define $\mathcal{V}$'s relations as Datalog programs over $\mathcal{R}$, what can we deduce about $\mathcal{R}$? Instead of using $\mathcal{M}$ directly, the approach we take is to "invert" $\mathcal{M}$ and use the inverse mapping $\mathcal{M}^{-1}$ instead. A well known trick to invert $\mathcal{M}$ consists of:

- First, "complete" all the IDB predicates in $\mathcal{M}$. For each IDB predicate v, consider all the rules in $\mathcal{M}$ using predicate v in the head, say $v(\overline{T}_k) :\Leftrightarrow B_k$ for $k = 1, \ldots, n$, where $\overline{T}_k$ denotes the arguments in the head. If $\overline{Z}_k$ denotes body variables that do not occur in the head, the formula

$$v(\overline{X}) \Leftrightarrow \bigcup_{k=1}^{n} (\exists \overline{Z}_k)(B_k \wedge \overline{X} = \overline{T}_k)$$

  expresses the assumption that v is completely defined; here, we are only interested in the forward implication. Please refer to [Clark78] for more details.

- Then, "skolemize" the existential variables. For each declaration of an existential variable, we introduce a new function symbol (a.k.a. Skolem function). Then each occurrence of that variable is replaced with a term with the new function and whose arguments are the free variables.

EXAMPLE 3.1 Consider Example 1.1 where we assume view V is completely defined.
Completing predicate v produces:

$$v(X, Y) \Leftrightarrow (\exists Z)\ [p_1(X, Z) \wedge p_2(Z, Y)]$$

Replacing the existential variable Z with a new (Skolem) function symbol g in the formula, we get:

$$v(X,Y) \Leftrightarrow p_1(X,g(X,Y)) \wedge p_2(g(X,Y),Y)$$

Using only the forward implication, we obtain the following rules for the inverse mapping $\mathcal{M}^{-1}$:

$$\begin{array}{lcl} p_1(X,g(X,Y)) & :- & v(X,Y). \\ p_2(g(X,Y),Y) & :- & v(X,Y). \end{array} \blacksquare$$

Conjunctive mappings

For the special case where the mapping $\mathcal{M}$ is conjunctive, each view predicate v in $\mathcal{V}$ is defined by a single rule in $\mathcal{M}$ of the form:

$$v(\overline{X}) \quad :- \quad G_1, \ldots, G_j, \ldots, G_n.$$

where each G_j is a subgoal with some $\mathcal{R}$ predicate and uses variables $\overline{X}_j$, and $\overline{X} \subseteq \bigcup_{j=1}^{n} \overline{X}_j$. For any predicate p from $\mathcal{R}$ that occurs in some subgoal G_j, completing v produces a rule of the form:

$$p(\overline{X}'_j) \quad :- \quad v(\overline{X}).$$

where $\overline{X}'_j$ is formed from $\overline{X}_j$ by replacing any variable u in $\overline{X}_j \Leftrightarrow \overline{X}$ with the term $f_u(X)$. The Skolem function f_u is a new symbol that depends only on v and u.

So inverse mappings $\mathcal{M}^{-1}$ for conjunctive views are very simple nonrecursive Horn programs whose rules have single goal bodies, and where Skolem functions only occur in rule heads.

Pseudo-equivalence for nonrecursive Datalog queries

A nonrecursive Datalog query Q (with predicate q) can be expressed as a collection of rules of the form:

$$q(\overline{X}) :- H_1, \ldots, H_j, \ldots, H_m.$$

where each H_j is a subgoal with some $\mathcal{R}$ predicate.

Let us define query Q' (with predicate q') to consist of rules of the form

$$q'(\overline{X}') :- R_1, \ldots, R_j, \ldots, R_m.$$

where each subgoal R_j, which uses some predicate v from $\mathcal{V}$, is the result of replacing H_j with the body of some rule $p(Y) :- v(Z)$ from $\mathcal{M}^{-1}$ when $p(Y)$ successfully unifies with H_j. If any function symbol remains in the resulting rule, the rule is ignored.

Proposition 3.1 *We claim that Q' is pseudo-equivalent to Q.* ∎

Proof: (Sketch) The last step in the construction of Q' essentially throws away all the answers to the query predicate that contain functional terms. While these answers represent those that depend on the actual relations in $\mathcal{R}$ (that are still constrained by the given $\mathcal{V}$ and mapping), the remaining answers that are retained correspond to those definite answers that logically follows from the given $\mathcal{V}$ and mapping (the largest set of answers that result from using any $\mathcal{R}$). ∎

EXAMPLE 3.2 Consider a query Q defined the rules $q(X) :- p_1(X,Y)$ and $q(Y) :- p_2(X,Y)$. Consider the view V from Example 3.1. The subgoal $p_1(X,Y)$ in the first rule for Q unifies with the head of $p_1(X',g(X',Y')) :- v(X',Y')$, and the corresponding rule for Q' becomes $q'(X') :- v(X',Y')$. Similarly, the rule for Q' that corresponds to the second rule for Q can be derived as $q'(Y') :- v(X',Y')$. ∎

Corollary 3.1 *Under conjunctive mappings, any nonrecursive Datalog query $Q(\mathcal{R})$ has a pseudo-equivalent query $Q'(\mathcal{V})$ in nonrecursive Datalog.* ▮

4 Solving the translation problem for conjunctive views

The previous section suggests a method to solve the translation problem

- For mappings $\mathcal{M}$ that define conjunctive views
- And for Q and Q' in recursive Datalog.

One of the key observations is to equate the set of answers for Q' that are deducible from $\mathcal{V}$, $\mathcal{M}$, and Q, with the least fixpoint of a new query Q^H [1] formed by combining Q with all the rules that result from completing the predicates in $\mathcal{V}$, that is

$$Q^H = Q \cup \mathcal{M}^{-1}$$

where Q^H uses q for its query predicate and the $\mathcal{V}$ predicates are the only EDB predicates (the EDB predicates in $\mathcal{M}$ are relabelled as IDB). In other words, Q^H, a Datalog program with function symbols, is treated as a query over $\mathcal{V}$.

Proposition 3.1 generalizes as follows.

Proposition 4.1 *Q^H computes the same function-free answers as Q'.* ▮

We have effectively reduced the problem of finding Datalog pseudo-equivalents of Q to that of finding Datalog queries Q' that have the same function-free least fixpoint as the Horn query Q^H for all database $\mathcal{V}$.

While Horn programs in general may not have a finite least fixpoint, the query Q^H in question here (i.e. constructed out of conjunctive views and Datalog queries) is only a special case of *term-bounded* queries, a larger class of Horn queries that do have finite answers.

We will show an algorithm that rewrites any given term-bounded Horn query over $\mathcal{V}$ to a Datalog query over $\mathcal{V}$ that produces the same function-free answers, for all databases $\mathcal{V}$.

Definition 4.1 (Term-bounded queries): A Horn query with function symbols is term-bounded if for every EDB, the size of all derivable tuples for every IDB relation is bounded. ▮

We assume the EDB relations contain no function symbols. The size of a tuple is the maximum height of the components of the tuple. These components may be functional terms whose leaves are constant symbols from the EDB and whose interior nodes are function symbols from the query. With this class of programs, unbounded growth of functional terms within derivable facts never happens for any given EDB. Term-boundedness is not a new concept and has been extensively studied in the literature [KRS88], [Sh87].

[1]The annotation "H" is used to remind us that the query is a Horn program whose function symbols are the result of skolemization.

$$\begin{tikzcd}
{<}q.j{>} \arrow[d, bend right=60, "0"'] & \text{Query } Q \\
{<}p.j{>} \arrow[u, bend right=60, "0"'] & \\
{<}v.j{>} \arrow[u, "0 \text{ or } 1"] & \text{Inverse Mapping } \mathcal{M}^{-1}
\end{tikzcd}$$

Figure 4: Syntactic check for term-boundedness of Q^H

Syntactic characterization

While the problem of determining if a Horn query is term-bounded is generally undecidable, there are many syntactic characterizations of Horn queries that only guarantee their term-boundedness but are not necessary. We will give a simple one here, with the understanding that there are others in the literature such as [KL88], [VG90], that can define larger classes of term-bounded Horn queries.

Consider a directed graph whose nodes represent components of predicates used in the program. An n-ary predicate p would be represented by n nodes denoted $<p.1>$, ... ,$<p.n>$ in the graph. For each rule

$$p(\ldots, T_j, \ldots) \text{ :- } \ldots, q(\ldots, U_i, \ldots), \ldots$$

where T_j denotes the term in the j^{th} component of the head and U_i the term in the i^{th} component of the subgoal with predicate q, there is an edge in the graph from $<Q.i>$ to $<P.j>$ if T_j and U_i share some variable. The edge is labeled "1" if there is a shared variable X such that the longest path from the root of T_j to any leaf X in T_j is greater than the longest path from the root of U_i to any leaf X in U_i. Otherwise, the edge is labelled "0".

The Horn query is term-bounded if the graph has no cycle with at least one edge labelled "1". Briefly, the reason is that when no such cycles are present, no component will be able to "feed" on itself.

EXAMPLE 4.1 Consider the general case of conjunctive views as presented in Section 2. Consider a Datalog query Q using q as the query predicate. The graph corresponding to Q^H is depicted in Figure 4.

In this graph, all edges in the lower band, which might be labelled "1", are only pointing upward, and all the potential cycles are confined to the upper band and have no edges labelled "1".

As a corollary, Horn queries formed by combining Datalog queries with conjunctive views are all term-bounded. ▮

Eliminating functional terms

Informally, the algorithm works on each rule all of whose variables are known to be bound to function-free symbols. The starting point obviously consists of those rules whose bodies only refer to EDB predicates. The exact function pattern of tuples each such rule can generate can precisely be predicted. The rule head is then "flattened", i.e. replaced with a "flat" head by means of a new predicate (called flat predicate) that represents the head's function pattern. All rules having subgoals unifiable with the head will have an arbitrary number of these subgoals replaced by flat predicate subgoals. This is reminiscent of a bottom-up symbolic evaluation where function patterns are propagated from the base predicates toward the query predicate. All function patterns that can potentially be generated are made explicit and encoded as flat predicates. At the end, all remaining rules that cannot be flattened are discarded. At the end, in order to remove answers that contain function symbols, those rules whose heads use flattened variations of the original query predicate that encode non trivial function patterns are discarded.

Algorithm 4.1 (Function symbol elimination by bottom up rewrite)

INPUT: A term-bounded Horn program with the query predicate *answer*.
OUTPUT: An equivalent Datalog program using no functional terms.
METHOD:

- Initialize INPUT-SET to the set of rules in the program, OUTPUT-SET to the empty set. Mark all EDB predicates "flat".
- While there is a rule in INPUT-SET all of whose subgoal predicates have been marked "flat", remove it from INPUT-SET, "flatten" it and put the result in OUTPUT-SET. This may cause new rules to be added to INPUT-SET.
- At the end, OUTPUT-SET contains the result. Its query predicate is the flat predicate that encodes the trivial function pattern (i.e. involving no function symbols) for *answer*.

To "flatten" a rule r with head $p(t_1, \ldots, t_n)$:

1. We first check if r's head is an instance of a function pattern that has been recorded so far.
2. If the check is negative, we record the new function pattern, create a new predicate p' (marked "flat") that becomes the representative for that pattern, and rewrite r's head in terms of p'. Note that the rewritten head is now flat, and is also recorded as a "flat head pattern". The arity of p' is the number of leaves in the tree that represents the function pattern the choice of which argument of p' corresponds to which leaf is not important as long as the correspondence is maintained). We continue in Step 4.
3. If the check is positive, we retrieve the predicate p' that is the representative for the existing function pattern. We then rewrite r's head in terms of p'. If the new head is subsumed by an existing flat head pattern, we are done flattening r. Otherwise we record the new flat head pattern and continue in Step 4.
4. For every rule in INPUT-SET that has p-subgoals in its body, create copies in each of which a different subset of the p-subgoals is selected to unify with r's head and rewrite in terms of p'. Note that some of these copies may be discarded because there is no variable substitution that makes all the p-subgoals in the subset unify with r's head. Variables that are used in subgoals with flat predicates are known to be bound to constant symbols and therefore cannot unify with functional terms. All the resulting rule copies are added back to INPUT-SET.

∎

Theorem 4.1 *If Algorithm 4.1 terminates, the initial INPUT-SET and the final OUTPUT-SET compute the same fixpoint.* ∎

Proof: (Sketch) We use induction on the number of "flatten" steps. Consider the following invariant: at each step, INPUT-SET $\cup$ OUTPUT-SET is equivalent to the original program. Note that equivalence here means that the two programs compute the same least fixpoint where tuples with function symbols are allowed.

Initially, the invariant trivially holds. Now we want to show that if the invariant holds before a "flatten" step, it still holds after. Let's denote $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) the program in INPUT-SET $\cup$ OUTPUT-SET before (resp. after) a "flatten" step. $\mathcal{P}_2$ cannot generate more tuples than $\mathcal{P}_1$, since new rules added to $\mathcal{P}_1$ are just instances of some of $\mathcal{P}_1$'s rules. $\mathcal{P}_2$ cannot generate less, since all possible invocations of the rule to be flattened are considered, each possibility resulting in a rule copy to be added to $\mathcal{P}_1$. Therefore, in terms of the original IDB predicates, $\mathcal{P}_1$ and $\mathcal{P}_2$ generate the same sets of tuples.

Assume the algorithm terminates. After the last "flatten" step, all rules in INPUT-SET cannot be flattened, i.e. they each contain a goal whose predicate is not flat. These rules are useless, since none can start generating tuples (for this, a rule must have all its subgoals with a flat predicate). Therefore, INPUT-SET can be dropped without affecting the program's answer. ▮

Theorem 4.2 *Algorithm 4.1 terminates.* ▮

Proof: (Sketch) There can only be a finite number of "flatten" steps. A step falls into one of the following cases:

1. A new flat predicate, encoding a new function pattern is created.
2. No new predicate is created but the new head is not subsumed by any existing flat head pattern.
3. No new predicate is created and the new head is subsumed by some existing flat head pattern.

First, we show there can only be a finite number of steps in Case 1. The key point is that the algorithm only generates function patterns that are possible. That is, for any function pattern generated, there is always an EDB extension that causes a tuple of the corresponding form to be generated (to see this, just consider the OUTPUT-SET, all rules in that set are well founded). Now, if there were an infinite number of function patterns generated, there must be an EDB extension that causes an infinite number of tuples to be generated. This is not possible since the input program is assumed to be term-bounded. So the number of steps in Case 1 is bounded. Second, for a given function pattern, there can only be a finite number of different flat head patterns consistent with the function pattern. Thus, the number of steps in Case 2 is also bounded. As a result, finally, the total number of rule copies the algorithm generates (by means of steps of the first two cases) is bounded, and therefore the number of steps in Case 3 is bounded. ▮

EXAMPLE 4.2 Consider the view from Example 1.1 and the following recursive Datalog query Q:

$$\begin{array}{lll} q(X,Y) & \text{:-} & p_1(X,Y). \\ q(X,Y) & \text{:-} & p_2(X,Y). \\ q(X,Y) & \text{:-} & q(X,Z),\ q(Z,Y). \end{array}$$

Algorithm 4.1 rewrites the corresponding Q^H to the following recursive Datalog query Q':

$$\begin{array}{lll} q'(X,Y) & \text{:-} & v(X,Y). \\ q'(X,Y) & \text{:-} & q'(X,Z),\ q'(Z,Y). \end{array}$$ ▮

Corollary 4.1 *For any term-bounded Horn query, there is a pure Datalog query that computes the same function-free answers.* ∎

Corollary 4.2 *For conjunctive views $\mathcal{V} = \mathcal{M}(\mathcal{R})$, any recursive Datalog query $Q(\mathcal{R})$ has a pseudo-equivalent query $Q'(\mathcal{V})$ in recursive Datalog.* ∎

5 Related work

One approach to query translation between semantically heterogeneous data sources consists of extending the data model to match the expressive power of the query language enriched to capture the more complex mappings. DeMichiel's work [DeM89] on extended relational model and operations exemplifies this approach where indefinite answers to queries are allowed.

In spirit, Qian's query mediation approach to semantic interoperation [Qian93] is related to ours, in which source and target queries are expressed in a traditional first order query language and are required to return definite answers. However, translatability issues and computational aspects of query reformulation are not addressed in her work.

Incomplete deductive databases have been studied in [Im86] and [KL88] but with an emphasis on query evaluation. In [KL88], Kifer et al. identified a class of Horn queries that can be efficiently evaluated which traditional top-down or bottom-up evaluation techniques would fail to terminate. In [Im86], Imielinski studied the complexity of processing different classes of Horn queries and presented both positive and negative results. While negative results in query evaluability can be clearly carried over to our translation problem, it is much less clear how positive results would apply here.

6 Discussion

Rewriting General Horn Queries

There are certain Horn queries that are not term-bounded but for which a top-down approach extending some of the ideas used in our algorithm (e.g. functional patterns in goals) appears to work well. An approach using top-down expansion to guide bottom-up rewrite probably deserves further study. However, the following result gives a lid on the classes of translatable Horn queries beyond which no translation algorithm can be found.

Theorem 6.1 *There is no algorithm that effectively translates any arbitrary Horn query into a Datalog query that has the same function-free fixpoint. The result remains negative even if we restrict ourselves to queries that contain only linear recursive rules with $\forall * \exists * \forall *$ prefixes.* ∎

Proof: (Sketch) This result is a straightforward consequence of the main undecidability result in [Im86] where it was shown that the derivation problem for conjunctive queries with linear recursive IDB rules with $\forall * \exists * \forall *$ prefixes is undecidable. ∎

General case of Datalog views

When views are allowed to be any Datalog query, completing the view predicates would result in a program that has the following features:

- Skolem terms for those body variables that are projected out in the original rules

- Disjunctions and literals involving equality in rule heads, when predicates in the original program are defined by multiple rules
- Recursion when the original program is recursive

As soon as we step beyond conjunctive views, that is, in considering a union of conjunctive queries (a fortiori recursive rules), we introduce disjunctive rule heads in the resulting mapping, which takes us outside the domain of Horn programs. Expanding rules of the form

$$H_1 \vee H_2 \vee \ldots \vee H_m \text{ :- } G_1, G_2, \ldots, G_n$$

into Horn rules with negation that is not of the simple kinds does not seem to get us closer to the solution.

Adding dependencies

Example 1.3 suggests using functional dependencies [Ull88] as a way to reduce uncertainties introduced by existential variables. Multivalued dependencies may also be used to achieve similar effects. In fact, if we apply this weaker form of dependencies to our example, we obtain a Horn program

$$\begin{array}{rcl} bookworld(I,T,f(I,T)) & \text{:-} & ref(I,T). \\ bookworld(I,g(I,C),C) & \text{:-} & publish(I,C). \\ bookworld(I,T1,P2) & \text{:-} & bookworld(I,T1,P1),\ bookworld(I,T2,P2). \end{array}$$

that will be rewritten into:

$$bookworld(I,T,P) \quad \text{:-} \quad ref(I,T), publish(I,P).$$

Nevertheless, this particular example does not show the added power of equality-generating dependencies that may produce many more possible term substitutions. This suggests a way to extend the unification used in our current translation algorithm to include the capability to perform term subsumptions.

Acknowledgments

We thank Prof. Jeff Ullman for his valuable comments regarding both technical contents and presentation of this material.

References

[CKPS95] S. Chaudhuri, R. Krishnamurthy, S. Potamianos and K Shim. *Optimizing queries with materialized views.* In Proceedings of the International Conference on Data Engineering, 1995.

[Chaw94] S. Chaw et al. *The TSIMMIS project: integration of heterogeneous information sources.* In IPSJ, Tokyo, Oct. 1994.

[Clark78] K. Clark. *Negation as failure.* In J. Minker and H. Gallaire, editors, Logic and Data Bases, Plenum Press, New York, 1978.

[DeM89] L. DeMichiel. *An approach to performing relational operations over mismatched domains.* IEEE Trans. on Knowledge and Data Engineering, Vol. 1 No. 4, Dec. 1989, pp. 485-493.

[DG96] O. M. Duschka and M. R. Genesereth. *Answering recursive queries using views.* Submitted for publication.

[Hu96] N. Q. Huyn. *Query reformulation under incomplete mappings.* Technical Report STAN–CS–TR–96–1576, Stanford University, Computer Science Department, 1996.

[Im86] T. Imielinski. *Complexity of query processing in the deductive databases with incomplete information.* Department of Computer Science, Rutgers University, DCS-TR-206, 1986.

[KL88] M. Kifer and E. L. Lozinskii. *SYGRAPH: Implementing logic programs in a database style.* IEEE Trans. on Software Engineering, vol. 14 No. 7, July 1988, pp. 922-935.

[KRS88] R. Krishnamurthy, R. Ramakrishnan and O. Shmueli. *A Framework for testing safety and effective computability of extended Datalog.* Proc. ACM Symp. on Management of Data, 1988, pp. 154-163.

[LMSS95] A. Levy, A. Mendelzon, Y. Sagiv and D. Srivastava. *Answering queries using views.* In Proc. ACM Symposium on Principles of Database Systems, San Jose, California, May 1995, pp 95-104.

[Qian93] X. Qian. *Semantic interoperation via intelligent mediation.* Proc. 3rd Int. Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, April 1993, pp. 228-231.

[Qian96] X. Qian. *Query folding.* In Proceedings of the 12th Int. Conf. on Data Engineering, 1996, pp 48-55.

[RSU95] A. Rajaraman, Y, Sagiv and J. Ullman. *Answering queries using templates with binding patterns.* In Proceedings of the ACM Symposium on Principles of Database Systems, 1995, pp 105-112.

[Sh87] O. Shmueli. *Decidability and expressiveness aspects of logic queries.* In Proc. 6th ACM Symp. on Principles of Database Systems, 1987, pp. 237-249.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, Vol. 1 and 2, Computer Science Press, Rockwille MD, 1988.

[VG90] A. Van Gelder. *Deriving constraints among argument sizes in logic programs.* In Proc. 9th ACM Symp. on Principles of Database Systems, 1990, pp. 47-60.

[Wie92] G. Wiederhold. *Mediators in the architecture of future information systems.* IEEE Computer, Vol. 25, No. 3, March 1992, pp. 38-49.