To appear in the Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Conference on Principles of Database Systems, PODS, Tucson, AZ, May 1997.

Answering Recursive Queries Using Views

Oliver M. Duschka
duschka@cs.stanford.edu
Department of Computer Science
Stanford University

Michael R. Genesereth
genesereth@cs.stanford.edu
Department of Computer Science
Stanford University

Abstract

We consider the problem of answering datalog queries using materialized views. The ability to answer queries using views is crucial in the context of information integration. Previous work on answering queries using views restricted queries to being conjunctive. We extend this work to general recursive queries: Given a datalog program $\mathcal{P}$ and a set of views, is it possible to find a datalog program that is equivalent to $\mathcal{P}$ and only uses views as EDB predicates? In this paper, we show that the problem of whether a datalog program can be rewritten into an equivalent program that only uses views is undecidable. On the other hand, we prove that a datalog program $\mathcal{P}$ can be effectively rewritten into a program that only uses views, that is contained in $\mathcal{P}$, and that contains all programs that only use views and are contained in $\mathcal{P}$. As a consequence, if there exists a program equivalent to $\mathcal{P}$ that only uses views, then our construction is guaranteed to yield a program equivalent to $\mathcal{P}$.

1 Introduction

We consider the problem of how to answer datalog programs if only materialized views are available as EDB predicates. This situation appears commonly in information integration, where information sources are considered to store materialized views over a global database schema (see [Ull97] for an excellent overview). Let us consider the datalog program

$$\begin{aligned}\mathcal{P}\colon\quad & q(X,Y) :- \ edge(X,Z),\ edge(Z,Y),\ black(Z) \\ & q(X,Y) :- \ edge(X,Z),\ black(Z),\ q(Z,Y)\end{aligned}$$

where *edge* and *black* are EDB predicates. If the predicate *edge* represents the edges of a graph, and the predicate *black* indicates which nodes are colored black, then the program computes the endpoints of paths of length at least two with black internal nodes. Only the two views

$$\begin{aligned}& v_1(X,Y) :- \ edge(X,Y),\ black(X) \\ & v_2(X,Y) :- \ edge(X,Y),\ black(Y)\end{aligned}$$

might be available. View v_1 stores edges with black starting nodes, and view v_2 stores edges with black end nodes. In this case, it is possible to rewrite datalog program $\mathcal{P}$ into an equivalent datalog program $\mathcal{P}_v$ that uses only views v_1 and v_2 as EDB predicates:

$$\begin{aligned}\mathcal{P}_v\colon\quad & q(X,Y) :- \ v_2(X,Z),\ v_1(Z,Y) \\ & q(X,Y) :- \ v_2(X,Z),\ q(Z,Y)\end{aligned}$$

The equivalence of $\mathcal{P}$ and $\mathcal{P}_v$ can be easily seen by expanding the view definitions in $\mathcal{P}_v$. However, if only view v_1 or only view v_2 were available, there wouldn't be an equivalent datalog program because every path would start or end with a black node respectively.

The first question to ask given a datalog program $\mathcal{P}$ and materialized views $v_1, \ldots, v_n$ is whether there is an equivalent datalog program that uses only $v_1, \ldots, v_n$ as EDB predicates. We show that this problem is undecidable, even if the views are conjunctive, and the datalog program and the views do not contain any built-in predicates.

Surprisingly, however, we are able to give a procedure to *construct* a datalog program which is the best possible rewriting. Formally, given a datalog program $\mathcal{P}$ and materialized conjunctive views $v_1, \ldots, v_n$, we show how to construct a datalog program $\mathcal{P}_v$ with the following properties:

(i) The only EDB predicates in $\mathcal{P}_v$ are the given views $v_1, \ldots, v_n$.

(ii) $\mathcal{P}_v$ is contained in $\mathcal{P}$.

(iii) Every datalog program $\mathcal{P}'_v$ that satisfies conditions (i) and (ii) is contained in $\mathcal{P}_v$.

As a consequence, we prove that our construction is guaranteed to yield an equivalent datalog program that only uses views as EDB predicates, whenever there exists such a program. When applied in the context of information integration, our construction yields query plans for using data available from information sources as much as possible, while still guaranteeing that the computed answers satisfy the user queries.

Example 1.1 Assume we want to integrate three databases that provide flight information. The first database stores pairs of cities between which Southwest Airlines has non-stop flights. The second database stores the cities that can be reached from Tucson by non-stop flights, together with the airlines offering these flights. The third database provides information on cities connected by United Airlines flights with one stop-over. We want to integrate these three databases so that users can ask arbitrary datalog queries over the EDB predicate $flight(From, To, Carrier)$. The

Copyright ©1997 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM Inc., fax +1 (212) 869-0481, or (permissions@acm.org)

intended meaning of $flight(tus, san, wn)$, for example, is that Southwest Airlines (wn) offers a non-stop flight from Tucson (tus) to San Diego (san). The three databases that we want to integrate can be seen as views over the predicate $flight$:

$$v_1(F,T) :- flight(F,T,wn)$$
$$v_2(T,C) :- flight(tus,T,C)$$
$$v_3(F,T) :- flight(F,Z,ua),\ flight(Z,T,ua)$$

Now assume a user is interested in the names of the cities that can be reached by plane from Tucson without changing airlines. The following is the corresponding user query:

$$c(F,T,C) :- flight(F,T,C)$$
$$c(F,T,C) :- flight(F,Z,C),\ c(Z,T,C)$$
$$q(T) :- c(tus,T,C)$$

The query defines a predicate c such that $c(san, sfo, ua)$, for example, means that there is a flight by United Airlines (ua) with possibly several stop-overs from San Diego to San Francisco (sfo). Using only the three databases that we have available, it is impossible to find all the cities that can be reached by plane from Tucson without changing airlines. The best that we can do is to find all the cities that can be reached from Tucson by flying Southwest Airlines (using v_1) or by flying United Airlines (using v_2 and v_3), together with all the cities that can be reached by plane from Tucson non-stop (using v_2). The construction that we will present in this paper yields a datalog program that in fact computes exactly these cities. □

It might seem that the possibility of effectively constructing maximally-contained datalog programs contradicts the previous undecidability result. There is no contradiction, however. Even if the constructed datalog program $\mathcal{P}_v$ is equivalent to the query datalog program $\mathcal{P}$, there is no way to determine this fact because it is the case that testing containment of $\mathcal{P}$ in $\mathcal{P}_v$ is undecidable.

2 Preliminaries

2.1 Datalog

A *Horn rule* is an expression of the form

$$p(\bar{X}) :- p_1(\bar{X}_1),\ \ldots,\ p_n(\bar{X}_n)$$

where p, and $p_1, \ldots, p_n$ are predicate names, and $\bar{X}$, $\bar{X}_1$, ..., $\bar{X}_n$ are tuples of variables, constants, and function terms. The *head* of the rule is $p(\bar{X})$, and its *body* is $p_1(\bar{X}_1)$, ..., $p_n(\bar{X}_n)$. Each $p_i(\bar{X}_i)$ is a *subgoal* of the rule. Every variable in the head of a rule must also occur in the body of the rule. A *logic program* is a set of Horn rules, and a *datalog program* is a set of function-free Horn rules. A predicate is an *intensional database predicate*, or *IDB predicate*, in a program $\mathcal{P}$ if it appears as the head of some rule in $\mathcal{P}$. Predicates not appearing in any head in $\mathcal{P}$ are *extensional database predicates*, or *EDB predicates*, in $\mathcal{P}$. We assume that every program has an IDB predicate q, called the query predicate, that represents the result of $\mathcal{P}$. Given a program, we can define a *dependency graph*, whose nodes are the predicate names appearing in the rules. There is an edge from the node of predicate p_i to the node of predicate p if p_i appears in the body of a rule whose head predicate is p. The program is *recursive* if there is a cycle in the dependency graph. A *conjunctive query* is a single non-recursive function-free Horn rule.

Example 2.1 Consider the datalog program

$$\mathcal{P}:\quad q(X,Y) :- edge(X,Z),\ edge(Z,Y),\ black(Z)$$
$$q(X,Y) :- edge(X,Z),\ black(Z),\ q(Z,Y)$$

from the introduction. Predicates $edge$ and $black$ are EDB predicates, and predicate q is an IDB predicate. The program is recursive because its dependency graph has a self-loop on predicate q. □

2.2 Containment

The input of a datalog program $\mathcal{P}$ consists of a database D storing instances of all EDB predicates in $\mathcal{P}$. Given such a database D, the output of $\mathcal{P}$, denoted $\mathcal{P}(D)$, is an instance of the query predicate q as determined by, for example, naive evaluation [Ull89]. A datalog program $\mathcal{P}'$ is *contained* in a datalog program $\mathcal{P}$ if, for all databases D, $\mathcal{P}'(D)$ is contained in $\mathcal{P}(D)$. Datalog programs $\mathcal{P}'$ and $\mathcal{P}$ are *equivalent* if $\mathcal{P}'$ and $\mathcal{P}$ are contained in one another. The problem of determining whether a datalog program $\mathcal{P}'$ is contained in a datalog program $\mathcal{P}$ is in general undecidable [Shm87]. The problem becomes decidable if $\mathcal{P}'$ is non-recursive. The following is a decision procedure for this containment proplem [RSUV89]. First, replace all variables in the non-recursive datalog program $\mathcal{P}'$ by distinct constants. Consider the database D_c that contains exactly the tuples corresponding to the subgoals in the "frozen" bodies of the rules in $\mathcal{P}'$. D_c is called the *canonical database* of $\mathcal{P}'$. Evaluate $\mathcal{P}$ on the canonical database. $\mathcal{P}'$ is contained in $\mathcal{P}$ if and only if the "frozen" heads of the rules in $\mathcal{P}'$ are contained in $\mathcal{P}(D_c)$.

Example 2.2 To determine whether the non-recursive datalog program

$$\mathcal{P}':\quad q(X,Y) :- edge(X,Z),\ edge(Z,Y),\ black(X),\ black(Z)$$
$$q(X,Y) :- edge(X,V),\ edge(V,W),\ edge(W,Y),\ black(V),\ black(W)$$

is contained in the datalog program $\mathcal{P}$ from example 2.1, we replace the variables in the two rules by distinct constants:

$$q(c_1,c_3) :- edge(c_1,c_2),\ edge(c_2,c_3),\ black(c_1),\ black(c_2)$$
$$q(c_4,c_7) :- edge(c_4,c_5),\ edge(c_5,c_6),\ edge(c_6,c_7),\ black(c_5),\ black(c_6)$$

This yields the following canonical database:

edge
$\langle c_1,c_2\rangle$, $\langle c_2,c_3\rangle$, $\langle c_4,c_5\rangle$, $\langle c_5,c_6\rangle$, $\langle c_6,c_7\rangle$

black
$\langle c_1\rangle$, $\langle c_2\rangle$, $\langle c_5\rangle$, $\langle c_6\rangle$

The output of datalog program $\mathcal{P}$ on the canonical database is $\langle c_1,c_3\rangle$, $\langle c_4,c_6\rangle$, $\langle c_5,c_7\rangle$, and $\langle c_4,c_7\rangle$. Because this output contains $\langle c_1,c_3\rangle$ and $\langle c_4,c_7\rangle$, $\mathcal{P}'$ is contained in $\mathcal{P}$. □

2.3 Retrievable Programs

A datalog program $\mathcal{P}_{v_1,\ldots,v_n}$ is *retrievable* if it does not contain any EDB predicates besides the view literals $v_1, \ldots, v_n$. We abbreviate $\mathcal{P}_{v_1,\ldots,v_n}$ to $\mathcal{P}_v$ in cases where the views $v_1, \ldots, v_n$ used are clear from the context. The *expansion* $\mathcal{P}_v^{exp}$ of a retrievable datalog program $\mathcal{P}_v$ is obtained from $\mathcal{P}_v$ by replacing all view literals with their definitions. Existentially quantified variables in views are replaced by new variables in the expansion.

Example 2.3 Given the view

$$v(X,Y) :- edge(X,Z),\ edge(Z,Y)$$

the datalog program

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

is retrievable because its only EDP predicate is the view literal v. The expansion of this retrievable datalog program is the following datalog program:

$$q(X,Y) :- edge(X,V),\ edge(V,Z),\ edge(Z,W),\ edge(W,Y)$$
$$q(X,Y) :- edge(X,V),\ edge(V,Z),\ q(Z,Y)$$

□

A retrievable datalog program P_v is contained in a datalog program $\mathcal{P}$ if P_v^{exp} is contained in $\mathcal{P}$. A retrievable program $\mathcal{P}_v$ is contained in another retrievable program $\mathcal{P}'_v$, if $\mathcal{P}_v^{exp}$ is contained in $(\mathcal{P}'_v)^{exp}$. A retrievable datalog program $\mathcal{P}_v$ is *maximally-contained* in a datalog program $\mathcal{P}$ if $\mathcal{P}_v$ is contained in $\mathcal{P}$, and for every retrievable datalog program $\mathcal{P}'_v$ that is contained in $\mathcal{P}$, $\mathcal{P}'_v$ is already contained in $\mathcal{P}_v$. Note that the notions of maximal containment and retrievability are relative to a fixed set of views.

3 Undecidability result

In this section we show the limits of reasoning about answering recursive queries using views. We prove that the question of the existence of an equivalent retrievable datalog program is undecidable. In comparison, this problem is in NP if queries are restricted to be conjunctive [LMSS95]. The undecidability proof uses a reduction from the containment problem of datalog programs, which is known to be undecidable [Shm87].

Theorem 3.1 *Given a datalog program $\mathcal{P}$ with EDB predicates $e_1,\dots,e_n$ and conjunctive views $v_1,\dots,v_m$ over predicates $e_1,\dots,e_n$, it is undecidable whether there is a retrievable datalog program $\mathcal{P}_v$ equivalent to $\mathcal{P}$.*

Proof. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two arbitrary datalog programs. We show that a decision procedure for the above problem would allow us to decide whether $\mathcal{P}_1$ is contained in $\mathcal{P}_2$. Because the containment problem for datalog programs is undecidable, this proves the claim. Without loss of generality we can assume that there are no IDB predicates with the same name in $\mathcal{P}_1$ and $\mathcal{P}_2$, and that the answer predicates in $\mathcal{P}_1$ and $\mathcal{P}_2$, named q_1 and q_2 respectively, have arity m. Let $\mathcal{P}$ be the datalog program consisting of all the rules in $\mathcal{P}_1$ and $\mathcal{P}_2$, and of the rules

$$q(X_1,\dots,X_m) :- q_1(X_1,\dots,X_m),\ e()$$
$$q(X_1,\dots,X_m) :- q_2(X_1,\dots,X_m)$$

where e is a new zero-ary EDB predicate. For every EDB predicate $e_i(X_1,\dots,X_{k_i})$ in $\mathcal{P}_1$ and $\mathcal{P}_2$ (but not for e) define the view

$$v_i(X_1,\dots,X_{k_i}) :- e_i(X_1\dots,X_{k_i}).$$

We show that $\mathcal{P}_1$ is contained in $\mathcal{P}_2$ if and only if there is a retrievable datalog program $\mathcal{P}_v$ equivalent to $\mathcal{P}$.

$''\Rightarrow''$: Assume $\mathcal{P}_1$ is contained in $\mathcal{P}_2$. Then $\mathcal{P}$ is equivalent to the program $\mathcal{P}_v$ consisting of all the rules of $\mathcal{P}_2$ with e_i's replaced by the corresponding v_i's, and the additional rule

$$q(X_1,\dots,X_m) :- q_2(X_1,\dots,X_m).$$

$''\Leftarrow''$: Assume there is a retrievable datalog program $\mathcal{P}_v$ equivalent to $\mathcal{P}$. Then for any instantiation of the EDB predicates, $\mathcal{P}$ and $\mathcal{P}_v$ yield the same result, especially for instantiations where e is the empty relation, and where e contains the empty tuple. If e is the empty relation then $\mathcal{P}$ produces exactly the tuples produced by $\mathcal{P}_2$, and therefore $\mathcal{P}_v$ does likewise. If e contains the empty tuple then $\mathcal{P}$ produces the union of the tuples produced by $\mathcal{P}_1$ and $\mathcal{P}_2$, and hence $\mathcal{P}_v$ produces this union. Because $\mathcal{P}_v$ does not contain e, $\mathcal{P}_v$ will produce the same set of tuples regardless of the instantiation of e. It follows that $\mathcal{P}_2$ is equivalent to the union of $\mathcal{P}_1$ and $\mathcal{P}_2$. Therefore, $\mathcal{P}_1$ is contained in $\mathcal{P}_2$. □

4 Construction of maximally-contained programs

As we will show in the following, it is possible to construct maximally-contained retrievable datalog programs. This is quite surprising, because in the case that there exists an equivalent retrievable datalog program, maximal containment implies equivalence. This is no contradiction to the undecidability result in the previous section, however. Even if we are able to construct an equivalent retrievable datalog program $\mathcal{P}_v$, it is still not possible to tell that $\mathcal{P}_v$ actually is equivalent to the original datalog program $\mathcal{P}$ because, in fact, determining whether $\mathcal{P}$ is contained in $\mathcal{P}_v$ is undecidable.

Our construction consists of two steps. In a first step we construct a program that might contain function symbols. The result of the first step is a retrievable logic program, but not necessarily a datalog program. The function symbols are introduced in a controlled fashion, so that in a second step we can transform the logic program into a datalog program. We then show that the resulting retrievable datalog program is maximally-contained in the query datalog program.

We need some notation for these transformations. The *inverse* v^{-1} of a view definition v is a set of rules having the view literal $v(X_1,\dots,X_m)$ as body, and each of the predicates in the body of v in turn as heads. The variables $X_1,\dots,X_m$ of the head of v remain unchanged. Every variable Y that appears in the body of v but not in the head, is consistently replaced by a term $f(X_1,\dots,X_m)$ where f is a function symbol, chosen anew for every existentially quantified variable. The inverse $\mathcal{V}^{-1}$ of a set of view definitions $\mathcal{V}$ is the union of the inverses v^{-1} of all view definitions v in $\mathcal{V}$.

Example 4.1 The inverse of the view definitions

$$v_1(X,Y) :- edge(X,Z),\ edge(Z,W),\ edge(W,Y)$$
$$v_2(X) :- edge(X,Z)$$

is the following set of rules:

$$edge(X,f_1(X,Y)) :- v_1(X,Y)$$
$$edge(f_1(X,Y),f_2(X,Y)) :- v_1(X,Y)$$
$$edge(f_2(X,Y),Y) :- v_1(X,Y)$$
$$edge(X,f_3(X)) :- v_2(X)$$

□

Given a datalog program $\mathcal{P}$ and a set of conjunctive views $\mathcal{V}$, the construction of the retrievable logic program is quite simple. We delete all rules from $\mathcal{P}$ that contain EDB predicates that do not appear in any of the view definitions. To

the resulting program, denoted as $\mathcal{P}^-$, we add the rules of $\mathcal{V}^{-1}$, and call the program so obtained $(\mathcal{P}^-, \mathcal{V}^{-1})$. Notice that the EDB predicates of the remaining rules of $\mathcal{P}$ are IDB predicates in $(\mathcal{P}^-, \mathcal{V}^{-1})$, because they appear in heads of the rules in $\mathcal{V}^{-1}$. Because naming of IDB predicates is arbitrary, one could rename the IDB predicates in $(\mathcal{P}^-, \mathcal{V}^{-1})$ so that their names differ from the names of the corresponding EDB predicates in $\mathcal{P}$. For ease of exposition, we will not do it here.

Example 4.2 Consider the datalog program

$$
\begin{array}{ll}
\mathcal{P}: & q(X,Y) :- \; edge(X,Y) \\
 & q(X,Y) :- \; edge(X,Z), \; q(Z,Y)
\end{array}
$$

which determines the transitive closure of the predicate *edge*. Assume there is only one materialized view available:

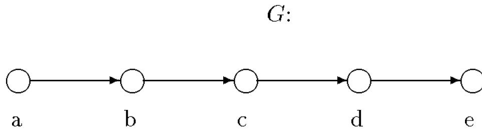
$$v(X,Y) :- \; edge(X,Z), \; edge(Z,Y)$$

View v stores endpoints of paths of length two. Just using this view, there is no way to determine the transitive closure of the predicate *edge*. The best one can hope to achieve is to compute the endpoints of paths of *even* lengths. Predicate *edge*, the only EDB predicate in $\mathcal{P}$, appears in the definition of v. Therefore, $(\mathcal{P}^-, \mathcal{V}^{-1})$ is just $\mathcal{P}$ with the rules of v^{-1} added:

$$
\begin{array}{lll}
(\mathcal{P}^-, \mathcal{V}^{-1}): & q(X,Y) & :- \; edge(X,Y) \\
 & q(X,Y) & :- \; edge(X,Z), \; q(Z,Y) \\
 & edge(X, f(X,Y)) & :- \; v(X,Y) \\
 & edge(f(X,Y), Y) & :- \; v(X,Y)
\end{array}
$$

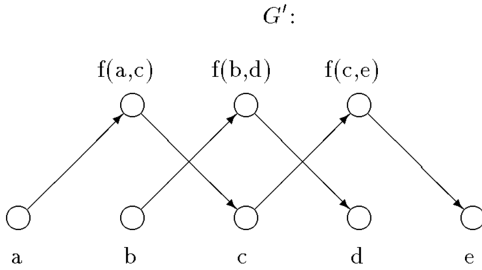
$(\mathcal{P}^-, \mathcal{V}^{-1})$ indeed yields all endpoints of paths of even length in its result. For example, assume that an instance of the EDB predicate *edge* in $\mathcal{P}$ represents the following graph:

G:



$(\mathcal{P}^-, \mathcal{V}^{-1})$ introduces three new constants, named $f(a,c)$, $f(b,d)$, and $f(c,e)$. The IDB predicate *edge* in $\mathcal{V}^{-1}$ represents the following graph:

G':



$\mathcal{P}^-$ computes the transitive closure of G'. Notice that the pairs in the transitive closure of G' that do not contain any of the new constants are exactly the endpoints of paths of even length in the original graph G. □

Bottom-up evaluation of logic programs is not guaranteed to terminate in general. This is because it might be possible to generate terms with arbitrary many function symbols. For example, bottom-up evaluation of the logic program

$$
\begin{array}{ll}
q(X) & :- \; p(X) \\
q(f(X)) & :- \; q(X)
\end{array}
$$

contains the infinite number of terms a, $f(a)$, $f(f(a))$, ... in its answer, if the EDB predicate p contains the constant a. In contrast, our construction produces logic programs whose bottom-up evaluation *is* guaranteed to terminate. The key observation is that function symbols are *only* introduced in inverse rules. Because inverse rules are not recursive, no terms with nested function symbols can be generated.

Theorem 4.1 *For every datalog program $\mathcal{P}$, every set of conjunctive views $\mathcal{V}$, and all finite instances of the views, the logic program $(\mathcal{P}^-, \mathcal{V}^{-1})$ has a unique finite minimal fixpoint. Furthermore, naive and semi-naive evaluation are guaranteed to terminate, and produce this unique fixpoint.*

Proof. (Sketch) $\mathcal{P}^-$ is recursive, but does not introduce function symbols. On the other hand, $\mathcal{V}^{-1}$ introduces function symbols, but is not recursive. Therefore, every bottom-up evaluation of $(\mathcal{P}^-, \mathcal{V}^{-1})$ will necessarily progress in two stages. In the first stage, the instances of the IDB predicates in $\mathcal{V}^{-1}$ are determined. The second stage will then be a standard datalog evaluation of $\mathcal{P}^-$. Because datalog programs have unique finite minimal fixpoints, this proves the claim. □

Given instances for its EDB predicates, a logic program might produce tuples containing function symbols in its result. Because instances of EDB predicates do not contain any function symbols, no datalog program produces tuples in its result containing function symbols. This motivates the definition of a filter that gets rid of all extraneous tuples. If D is a database containing instances of the EDB predicates of a logic program $\mathcal{P}$, then let $\mathcal{P}(D)\downarrow$ be the set of all tuples in $\mathcal{P}(D)$ that do not contain function symbols. Let $\mathcal{P}\downarrow$ be the program that given a database D computes $\mathcal{P}(D)\downarrow$.

The following theorem shows that the simple construction of adding rules of the inverses of the view definitions yields a logic program that uses the views in the best possible way. After discarding all tuples containing function symbols, the result of $(\mathcal{P}^-, \mathcal{V}^{-1})$ is contained in $\mathcal{P}$. Moreover, the result of every retrievable datalog program that is contained in $\mathcal{P}$ is already contained in $(\mathcal{P}^-, \mathcal{V}^{-1})$.

Theorem 4.2 *For every datalog program $\mathcal{P}$ and every set of conjunctive views $\mathcal{V}$, $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ is maximally-contained in $\mathcal{P}$. Moreover, $(\mathcal{P}^-, \mathcal{V}^{-1})$ can be constructed in time polynomial in the size of $\mathcal{P}$ and $\mathcal{V}$.*

Proof. First we prove that $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ is contained in $\mathcal{P}$. Let $E_1, \ldots, E_n$ be instances of the EDB predicates in $\mathcal{P}$. $E_1, \ldots, E_m$ determine the instances of the views in $\mathcal{V}$ which in turn are the EDB predicates of $(\mathcal{P}^-, \mathcal{V}^{-1})$. Assume that $(\mathcal{P}^-, \mathcal{V}^{-1})$ produces a tuple t that does not contain any function symbols. Consider the derivation tree of t in $(\mathcal{P}^-, \mathcal{V}^{-1})$. All the leaves are view literals because view literals are the only EDB predicates of $(\mathcal{P}^-, \mathcal{V}^{-1})$. Removing all leaves from this tree produces a tree with the original EDB predicates from $\mathcal{P}$ as new leaves. Because the instances of the views are derived from $E_1, \ldots, E_n$, there are constants in $E_1, \ldots, E_n$ such that consistently replacing function terms with these constants yields a derivation tree of t in $\mathcal{P}$. Therefore, $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ is contained in $\mathcal{P}$.

Let $\mathcal{P}_v$ be an arbitrary retrievable datalog program contained in $\mathcal{P}$. We have to prove that $\mathcal{P}_v$ is also contained in $(\mathcal{P}^-, \mathcal{V}^{-1})$. Let c_v be an arbitrary conjunctive query generated by $\mathcal{P}_v$. If we can prove that c_v^{exp} is contained in

$(\mathcal{P}^-, \mathcal{V}^{-1})$, then $\mathcal{P}_v$ is contained in $(\mathcal{P}^-, \mathcal{V}^{-1})$, which proves the claim. Let D_c be the canonical database of c_v^{exp}. Because c_v^{exp} is contained in $\mathcal{P}$, $c_v^{exp}(D_c)$ is contained in the output of $\mathcal{P}$ applied to D_c. Let c be the conjunctive query generated by $\mathcal{P}$ that produces $c_v^{exp}(D_c)$. Because all predicates of query c are also in c_v^{exp}, and all predicates in c_v^{exp} appear in some view definition, c is also generated by $\mathcal{P}^-$. Because c_v^{exp} is contained in c, there is a containment mapping h from c to c_v^{exp} [CM77]. Every variable Z in c_v^{exp} that does not appear in c_v is existentially quantified in some view $v_i(X_1, \ldots, X_m)$ in c_v. Let k be the mapping that maps every such variable Z to the corresponding term $f(X_1, \ldots, X_m)$ used in v_i^{-1}. Because $\mathcal{P}^-$ can derive c, $\mathcal{P}^-$ can also derive the more specialized conjunctive query $k(h(c))$. Using rules in $\mathcal{V}^{-1}$, the derivation of $k(h(c))$ in $\mathcal{P}^-$ can be extended to a derivation of a conjunctive query c' that only contains view literals. The identity mapping is a containment mapping from c' to c_v. This proves that $\mathcal{P}_v$ is contained in $(\mathcal{P}^-, \mathcal{V}^{-1})$.

$(\mathcal{P}^-, \mathcal{V}^{-1})$ can be constructed in time polynomial in the size of $\mathcal{P}$ and $\mathcal{V}$, because every subgoal in a view definition in $\mathcal{V}$ corresponds to exactly one inverse rule in $\mathcal{V}^{-1}$. □

5 Eliminating function symbols

Although in section 4 we demonstrated an efficient procedure to answer a query datalog program as well as possible given only materialized views, it is desirable to transform the constructed logic program to a datalog program that represents this answer. This means that we are looking for a datalog program that is equivalent to $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$. As will be seen, this is not difficult. The key observation is that there are only finitely many function symbols in $(\mathcal{P}^-, \mathcal{V}^{-1})$. Because nested function expressions can never be generated using bottom-up evaluation, it is possible, with a little bit of bureaucracy, to keep track of function terms produced by $(\mathcal{P}^-, \mathcal{V}^{-1})$ without actually generating tuples containing function terms.

The transformation proceeds in a bottom up fashion. Function terms like $f(X_1, \ldots, X_k)$ in the IDB predicates of $\mathcal{V}^{-1}$ are eliminated by replacing them by the list of variables $X_1, \ldots, X_k$ that occur in them. The IDB predicate names need to be annotated to indicate that $X_1, \ldots, X_k$ belonged to the function term $f(X_1, \ldots, X_k)$. For instance, in Example 4.2 the rule

$$edge(X, f(X,Y)) :- v(X,Y)$$

is replaced by the rule

$$edge^{\langle 1, f(2,3)\rangle}(X,X,Y) :- v(X,Y)$$

The annotation $\langle 1, f(2,3)\rangle$ represents the fact that the first argument in $edge^{\langle 1, f(2,3)\rangle}$ is identical to the first argument in $edge$, and that the second and third argument in $edge^{\langle 1, f(2,3)\rangle}$ combine to a function term with the function symbol f as the second argument of $edge$. If bottom-up evaluation of $(\mathcal{P}^-, \mathcal{V}^{-1})$ can yield a function term for an argument of an IDB predicate in $\mathcal{P}^-$, then a new rule is added with correspondingly expanded and annotated predicates. The following example shows this transformation.

Example 5.1 The logic program from example 4.2 is transformed to the following datalog program. The lines indicate the stages in the generation of the datalog rules.

$$edge^{\langle 1, f(2,3)\rangle}(X,X,Y) :- v(X,Y)$$
$$edge^{\langle f(1,2),3\rangle}(X,Y,Y) :- v(X,Y)$$

$$q^{\langle 1, f(2,3)\rangle}(X,Y_1,Y_2) :- edge^{\langle 1, f(2,3)\rangle}(X,Y_1,Y_2)$$
$$q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) :- edge^{\langle f(1,2),3\rangle}(X_1,X_2,Y)$$

$$q(X,Y) :- edge^{\langle 1, f(2,3)\rangle}(X,Z_1,Z_2), q^{\langle f(1,2),3\rangle}(Z_1,Z_2,Y)$$
$$q^{\langle f(1,2), f(3,4)\rangle}(X_1,X_2,Y_1,Y_2) :- edge^{\langle f(1,2),3\rangle}(X_1,X_2,Z), q^{\langle 1, f(2,3)\rangle}(Z,Y_1,Y_2)$$

$$q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) :- edge^{\langle f(1,2),3\rangle}(X_1,X_2,Z), q(Z,Y)$$
$$q^{\langle 1, f(2,3)\rangle}(X,Y_1,Y_2) :- edge^{\langle 1, f(2,3)\rangle}(X,Z_1,Z_2), q^{\langle f(1,2), f(3,4)\rangle}(Z_1,Z_2,Y_1,Y_2)$$

□

The generated datalog programs show explicitly in which arguments the original logic program was able to produce function terms. Some tuples with function symbols might never have been able to contribute to an answer without function symbols. Using our exact bookkeeping of function symbols, we are able to eliminate the derivations of these useless tuples. In the following we are going to present two optimizations.

Define a predicate p to be *relevant* if there is a path in the dependency graph from p to the query predicate q. If a predicate p is not relevant, then no derivation of a tuple in the answer requires p. Therefore, all rules for irrelevant predicates can be dropped without losing any answers.

Example 5.2 In the dependency graph for the datalog program in example 5.1, there are no paths from predicates $q^{\langle 1, f(2,3)\rangle}$ and $q^{\langle f(1,2), f(3,4)\rangle}$ to q. Therefore, these two predicates are irrelevant. The three rules defining the irrelevant predicates can be dropped. The following is the datalog program after the first optimization step:

$$edge^{\langle 1, f(2,3)\rangle}(X,X,Y) :- v(X,Y)$$
$$edge^{\langle f(1,2),3\rangle}(X,Y,Y) :- v(X,Y)$$
$$q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) :- edge^{\langle f(1,2),3\rangle}(X_1,X_2,Y)$$
$$q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) :- edge^{\langle f(1,2),3\rangle}(X_1,X_2,Z), q(Z,Y)$$
$$q(X,Y) :- edge^{\langle 1, f(2,3)\rangle}(X,Z_1,Z_2), q^{\langle f(1,2),3\rangle}(Z_1,Z_2,Y)$$

□

The second optimization doesn't reduce the number of derivations, but is an easy way to save unneccessary copying of data during the evaluation of the datalog progam. If p is a predicate in a datalog program that has only one rule, and the body of this rule has only one subgoal, then predicate p can be eliminated from the program. For every rule having

p as one of its subgoals, unify this subgoal with the head of the rule of p, and replace the subgoal by the corresponding body of the rule of p.

Example 5.3 Predicates $edge^{\langle 1,f(2,3)\rangle}$ and $edge^{\langle f(1,2),3\rangle}$ in example 5.1 have only one rule and only one subgoal in the bodies of their rules, and can therefore be eliminated. The following is the resulting datalog program:

$$q^{\langle f(1,2),3\rangle}(X,Y,Y) \ :- \ v(X,Y)$$
$$q^{\langle f(1,2),3\rangle}(X,Z,Y) \ :- \ v(X,Z), \ q(Z,Y)$$
$$q(X,Y) \ :- \ v(X,Z), \ q^{\langle f(1,2),3\rangle}(X,Z,Y)$$

□

In the following, let us denote the resulting datalog program by $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$. Clearly, there is a one-to-one correspondence between bottom-up evaluations in $(\mathcal{P}^-,\mathcal{V}^{-1})$ and in $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$. Because we keep track of function symbols in $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$, we know that the resulting instance of the query predicate q is exactly the subset of the result of $(\mathcal{P}^-,\mathcal{V}^{-1})$ that does not contain function symbols.

Theorem 5.1 *For every datalog program $\mathcal{P}$ and every set of conjunctive views $\mathcal{V}$, the program $(\mathcal{P}^-,\mathcal{V}^{-1})\downarrow$ is equivalent to $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$.*

Assume for a datalog program $\mathcal{P}$ and a set of conjunctive views $\mathcal{V}$ there exists a retrievable datalog program $\mathcal{P}_v$ that is equivalent to $\mathcal{P}$. Because of theorems 4.2 and 5.1, we know that the retrievable datalog program $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$ is maximally-contained in $\mathcal{P}$. This implies that $\mathcal{P}_v$ is contained in $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$, and therefore $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$ is equivalent to $\mathcal{P}$.

Corollary 5.1 *For every datalog program $\mathcal{P}$ and every set of conjunctive views $\mathcal{V}$ over the EDB predicates of $\mathcal{P}$, the retrievable datalog program $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$ is maximally-contained in $\mathcal{P}$. Moreover, if there exists a retrievable datalog program that is equivalent to $\mathcal{P}$, then $(\mathcal{P}^-,\mathcal{V}^{-1})^{datalog}$ is equivalent to $\mathcal{P}$.*

6 Related work

The question of answering queries using views has attracted a lot of interest recently because of its application in information integration [Ull97, LRO96a, LRO96b, DG97] and query optimization [CKPS95, YL87]. All work so far has been restricted to the case of *conjunctive* queries. Levy et al. [LMSS95] showed that the question of determining whether a conjunctive query can be rewritten to an equivalent conjunctive query that only uses views is NP-complete. Rajaraman et al. extended this work in [RSU95] to include binding patterns in view definitions. Binding patterns can be used to express restricted query capabilities of information sources. In order to model information sources with more complex query capabilities, Levy et al. [LRU96] considered how to answer queries given an infinite number of conjunctive views. Huyn [Huy96] proposed "pseudo-equivalent" rewritings in the case that no equivalent rewritings exist. These ideas were used in [Qia96] to give an algorithm for rewriting conjunctive queries given materialized views. Our work is the first that extends the work on answering queries using views to *recursive* queries.

6.1 Comparison with other algorithms

In the following, we are going to compare the construction presented in this paper with two other algorithms for answering queries using views: the *bucket algorithm* [LRO96a, LRO96b], and the *unification-join algorithm* [Qia96]. Because these algorithms cannot handle recursive queries, we will illustrate the differences between our construction and these two algorithms using a non-recursive example query.

Example 6.1 As a variation on example 1.1 assume that three databases are available which are described by the following views:

$$v_1(F,T) \ :- \ flight(F,T,wn)$$
$$v_2(F,T) \ :- \ flight(F,T,ua)$$
$$v_3(F,T,C) :- \ flight(F,Z,C), \ flight(Z,T,C)$$

The first and second database store the pairs of cities between which Southwest Airlines (wn) and United Airlines (ua) respectively offer direct flights. The third database stores pairs of cities that are connected by flights with one stop-over, together with the airlines that offer these flights. As an example query, assume a user wants to know the airlines that fly from Tucson to San Francisco with at most one stop-over:

$$q(C) \ :- \ flight(tus,sfo,C) \qquad (\alpha)$$
$$q(C) \ :- \ flight(tus,Z,C), \ flight(Z,sfo,C) \qquad (\beta)$$

Using the construction in section 4, the following maximally-contained logic program can be obtained in polynomial time:

$$flight(F,T,wn) :- \ v_1(F,T)$$
$$flight(F,T,ua) \ :- \ v_2(F,T)$$
$$flight(F,g(F,T,C),C) :- \ v_3(F,T,C) \qquad (*)$$
$$flight(g(F,T,C),T,C) :- \ v_3(F,T,C) \qquad (*)$$
$$q(C) \ :- \ flight(tus,sfo,C)$$
$$q(C) \ :- \ flight(tus,Z,C), \ flight(Z,sfo,C)$$

The transformation presented in section 5 would remove the two rules marked with (*), and would add the following rule:

$$q(C) \ :- \ v_3(F,T,C)$$

□

6.1.1 Bucket algorithm

The bucket algorithm is the algorithm used for query planning in the Information Manifold system [KLSS95, LRO96a, LRO96b]. For each subgoal p_i in the user query, a "bucket" B_i is created. If v_j is a view containing a predicate r unifiable with p_i, then $v_j\sigma$ is inserted into B_i, where σ is a most general unifier of p_i and r prefering the variables in p_i. For each conjunctive user query c separately, the bucket algorithm constructs retrievable conjunctive queries with the same head as c, and all possible combinations of view literals taken from the buckets corresponding to the subgoals of c as bodies. For each of these retrievable queries, the algorithm checks whether it can add a constraint C to the body, such that the expansion of the resulting query is contained in c. All retrievable conjunctive queries that pass this containment test, will be evaluated to find the answer to the user query.

Example 6.2 Applied to the query in example 6.1, the bucket algorithm creates three buckets B_1, B_2, and B_3 for the three subgoals $flight(tus,sfo,C)$, $flight(tus,Z,C)$, and $flight(Z,sfo,C)$ respectively. The buckets are filled as follows:

$\underline{B_1}$	$\underline{B_2}$	$\underline{B_3}$
$v_1(tus, sfo)$	$v_1(tus, Z)$	$v_1(Z, sfo)$
$v_2(tus, sfo)$	$v_2(tus, Z)$	$v_2(Z, sfo)$
$v_3(tus, T_1, C)$	$v_3(tus, T_2, C)$	$v_3(Z, T_3, C)$
$v_3(F_1, sfo, C)$	$v_3(F_2, Z, C)$	$v_3(F_3, sfo, C)$

For each of the four retrievable queries

$$
\begin{array}{ll}
q(C) :- v_1(tus, sfo) & (1) \\
q(C) :- v_2(tus, sfo) & (2) \\
q(C) :- v_3(tus, T_1, C) & \\
q(C) :- v_3(F_1, sfo, C) &
\end{array}
$$

the algorithm checks whether, after adding some constraints, its expansion is contained in the conjunctive user query (α). Further, the following sixteen retrievable queries are checked to see whether, after adding some constraints, their expansion is contained in the conjunctive user query (β):

$$
\begin{array}{ll}
q(C) :- v_1(tus, Z),\ v_1(Z, sfo) & (3) \\
q(C) :- v_1(tus, Z),\ v_2(Z, sfo) & \\
q(C) :- v_1(tus, Z),\ v_3(Z, T_3, C) & (4) \\
q(C) :- v_1(tus, Z),\ v_3(F_3, sfo, C) & (5) \\
q(C) :- v_2(tus, Z),\ v_1(Z, sfo) & \\
q(C) :- v_2(tus, Z),\ v_2(Z, sfo) & (6) \\
q(C) :- v_2(tus, Z),\ v_3(Z, T_3, C) & (7) \\
q(C) :- v_2(tus, Z),\ v_3(F_3, sfo, C) & (8) \\
q(C) :- v_3(tus, T_2, C),\ v_1(Z, sfo) & \\
q(C) :- v_3(tus, T_2, C),\ v_2(Z, sfo) & \\
q(C) :- v_3(tus, T_2, C),\ v_3(Z, T_3, C) & (9) \\
q(C) :- v_3(tus, T_2, C),\ v_3(F_3, sfo, C) & (10) \\
q(C) :- v_3(F_2, Z, C),\ v_1(Z, sfo) & (11) \\
q(C) :- v_3(F_2, Z, C),\ v_2(Z, sfo) & (12) \\
q(C) :- v_3(F_2, Z, C),\ v_3(Z, T_3, C) & (13) \\
q(C) :- v_3(F_2, Z, C),\ v_3(F_3, sfo, C) & (14)
\end{array}
$$

For each numbered retrievable query, a constraint can be added to its body such that it passes the containment test. For example, the constraint that needs to be added to query (1) is $''C = wn''$, and the constraint that needs to be added to query (10) is $''T_2 = sfo''$. □

As the example shows, the bucket algorithm has to perform a lot of containment tests. This is quite expensive, especially because testing containment of conjunctive queries is NP-complete.

6.1.2 Unification-join algorithm

The first step of the unification-join algorithm is the same as the first step of the construction given in this paper, namely the generation of inverse rules. However, whereas our construction transforms the original query together with the inverse rules into a retrievable datalog program, the unification-join algorithm constructs a set of retrievable conjunctive queries using the so-called unification-join as a central step.

For each subgoal p_i in the user query, a "label" L_i is created. If $r :- v$ is one of the inverse rules, and r and p_i are unifiable, then the pair $(\sigma \downarrow p_i,\ v\sigma)$ is inserted into L_i provided that $\sigma \downarrow q$ does not contain any function terms. Here, σ is a most general unifier of p_i and r, and $\sigma \downarrow p_i$ and $\sigma \downarrow q$ are the restriction of σ to the variables in p_i and to the variables in the query predicate q respectively. The unification-join of two labels L_1 and L_2, denoted $L_1 \overset{u}{\bowtie} L_2$, is defined as follows.

If L_1 contains a pair (σ_1, t_1) and L_2 contains a pair (σ_2, t_2), then $L_1 \overset{u}{\bowtie} L_2$ contains the pair $(\sigma_1\sigma \cup \sigma_2\sigma, (t_1 \wedge t_2)\sigma)$ where σ is a most general substitution such that $\sigma_1\sigma \downarrow \sigma_2 = \sigma_2\sigma \downarrow \sigma_1$, provided there is such a substitution σ, and provided $\sigma_1\sigma \downarrow q$, $\sigma_2\sigma \downarrow q$, and $(t_1 \wedge t_2)\sigma$ do not contain any function terms. If $(\sigma, v_{i_1} \wedge \ldots \wedge v_{i_n})$ is in the unification-join of all labels corresponding to the subgoals in one of the conjunctive user queries, and this user query has head h, then the retrievable query with head $h\sigma$ and body $v_{i_1}, \ldots, v_{i_n}$ is part of the result.

Example 6.3 Applied to the query in example 6.1, the unification-join algorithm generates three labels corresponding to the subgoals $flight(tus, sfo, C)$, $flight(tus, Z, C)$, and $flight(Z, sfo, C)$ respectively:

$\underline{L_1}$
$(\{C \rightarrow wn\},\ v_1(tus, sfo))$
$(\{C \rightarrow ua\},\ v_2(tus, sfo))$

$\underline{L_2}$
$(\{C \rightarrow wn\},\ v_1(tus, Z))$
$(\{C \rightarrow ua\},\ v_2(tus, Z))$
$(\{Z \rightarrow g(tus, T, C)\},\ v_3(tus, T, C))$

$\underline{L_3}$
$(\{C \rightarrow wn\},\ v_1(Z, sfo))$
$(\{C \rightarrow ua\},\ v_2(Z, tus))$
$(\{Z \rightarrow g(F, sfo, C)\},\ v_3(F, sfo, C))$

The unification-join of L_2 and L_3 is

$\underline{L_2 \overset{u}{\bowtie} L_3}$
$(\{C \rightarrow wn\},\ v_1(tus, Z) \wedge v_1(Z, sfo))$
$(\{C \rightarrow ua\},\ v_2(tus, Z) \wedge v_2(Z, sfo))$
$(\{Z \rightarrow g(tus, sfo, C)\},\ v_3(tus, sfo, C))$.

The labels corresponding to conjunctive queries (α) and (β) are L_1 and $L_2 \overset{u}{\bowtie} L_3$ respectively. The retrievable conjunctive queries that can be constructed from L_1 and $L_2 \overset{u}{\bowtie} L_3$ are:

$$
\begin{array}{ll}
q(wn) & :- v_1(tus, sfo) \\
q(ua) & :- v_2(tus, sfo) \\
q(wn) & :- v_1(tus, Z),\ v_1(Z, sfo) \\
q(ua) & :- v_2(tus, Z),\ v_2(Z, sfo) \\
q(C) & :- v_3(tus, sfo, C)
\end{array}
$$

□

The unification-join algorithm doesn't require any containment tests. However, it might generate an exponential number of conjunctive queries in cases when our algorithm generates a small datalog program. As an example, assume that there are k views of the form

$$v_i(X, Y) :- p(X, Y),\ p_i(X, Y) \quad \text{for } i = 1, \ldots, k.$$

Given the user query

$$q(X_0, X_n) :- p(X_0, X_1),\ p(X_1, X_2),\ \ldots,\ p(X_{n-1}, X_n)$$

the constructed maximally-contained datalog program is the following:

$$
\begin{array}{ll}
p(X, Y) & :- v_1(X, Y) \\
 & \vdots \\
p(X, Y) & :- v_k(X, Y) \\
q(X_0, X_n) & :- p(X_0, X_1),\ p(X_1, X_2),\ \ldots,\ p(X_{n-1}, X_n)
\end{array}
$$

Evaluating this datalog program requires $k-1$ unions and $n-1$ joins. On the other hand, the unification-join algorithm yields the following k^n retrievable conjunctive queries:

$$q(X_0, X_n) :- v_{j_1}(X_0, X_1),\ v_{j_2}(X_1, X_2),\ \ldots,\ v_{j_n}(X_{n-1}, X_n) \quad \text{for all } j_1, \ldots, j_n \in \{1, \ldots, k\}.$$

Evaluating these conjunctive queries requires $k^n - 1$ unions and $(n-1)k^n$ joins.

7 Conclusions and future work

This paper considered the problem of answering datalog queries if only materialized views are available as EDB predicates. We showed that for recursive queries and conjunctive views it is undecidable whether there exists an equivalent retrievable datalog program. For all practical purposes the much more important problem is to construct retrievable programs that can use the available views in a best possible manner, i.e. to construct maximally-contained programs. We showed that a simple, polynomial-time construction yields maximally-contained logic programs. Finally, we demonstrated how to translate these logic programs into pure datalog programs.

It would be interesting to investigate whether there are classes of datalog programs for which the problem of finding an equivalent datalog program that only uses views becomes decidable. The construction in section 4 can be extended to take functional dependencies and restrictions on binding-patterns into account. It should also be possible to incorporate built-in order predicates. Views in this paper were restricted to conjunctive queries. We are working on generalizing the ideas presented here to views that have disjunctions in their description. Finally, it is an open question whether view definitions can themselves be recursive.

Acknowledgments

We would like to thank Whitney Carrico and Harish Devarajan for helpful comments that improved this paper.

References

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseak Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*, IEEE Comput. Soc. Press, pages 190–200, Los Alamitos, CA, 1995.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on the Theory of Computing*, pages 77–90, 1977.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Query planning in Infomaster. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, San Jose, CA, 1997.

[Huy96] Nam Quan Huyn. A more aggressive use of views to extract information. Technical Report STAN-CS-TR-96-1577, Department of Computer Science, Stanford University, 1996.

[KLSS95] Thomas Kirk, Alon T. Levy, Yehoshua Sagiv, and Divesh Srivastava. The Information Manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Stanford, CA, 1995.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Divesh Srivastava, and Yehoshua Sagiv. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, CA, 1995.

[LRO96a] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI-96*, Portland, OR, 1996.

[LRO96b] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases*, Bombay, India, 1996.

[LRU96] Alon Y. Levy, Anand Rajaraman, and Jeffrey D. Ullman. Answering queries using limited external processors. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Montreal, Canada, 1996.

[Qia96] Xiaolei Qian. Query folding. In *Proceedings of the 12th International Conference on Data Engineering*, pages 48–55, New Orleans, LA, 1996.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1995.

[RSUV89] Raghu Ramakrishnan, Yehoshua Sagiv, Jeffrey D. Ullman, and Moshe Y. Vardi. Proof-tree transformation theorems and their applications. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Princicples of Database Systems*, pages 172 – 181, Philadelphia, PA, 1989.

[Shm87] Oded Shmueli. Decidability and expressiveness aspects of logic queries. In *Proceedings of the Sixth ACM Symposium on Principles of Database Systems*, pages 237 – 249, 1987.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledgebase Systems*, volume 2. Computer Science Press, 1989.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory*, 1997.

[YL87] H. Z. Yang and P.-Å. Larson. Query transformation for PSJ-queries. In *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, pages 245–254, Los Altos, CA, 1987.