Query Containment for Data Integration Systems [*]

Todd Millstein
University of Washington
Seattle, Washington
todd@cs.washington.edu

Alon Levy
University of Washington
Seattle, Washington
alon@cs.washington.edu

Marc Friedman[*]
Viathan Corporation
Seattle, Washington
marc@viathan.com

ABSTRACT

The problem of query containment is fundamental to many aspects of database systems, including query optimization, determining independence of queries from updates, and rewriting queries using views. In the data integration framework, however, the standard notion of query containment does not suffice. We define *relative containment*, which formalizes the notion of query containment relative to the sources available to the integration system. First we provide optimal bounds for relative containment for several important classes of datalog queries, including the common case of conjunctive queries. Next we provide bounds for the case when sources enforce access restrictions in the form of binding pattern constraints. Surprisingly, we show that relative containment for conjunctive queries is still decidable in this case, even though it is known that finding all answers to such queries may require a recursive datalog program over the sources. Finally, we provide tight bounds for variants of relative containment when the queries and source descriptions may contain comparison predicates.

1. INTRODUCTION

Data integration systems provide users a uniform interface to a multitude of data sources. Prime examples of data integration applications include querying sources on the WWW, querying multiple databases within an enterprise and querying disparate parts of a large-scale scientific experiment. A data integration system frees its users from having to locate the sources relevant to their query, interact with each source in isolation, and manually combine the data from the different sources. The problem of data integration has already fueled significant research [26; 13; 4; 5; 20; 17; 22; 25; 32; 18; 2] as well as several industrial solutions.

[*]This research was funded by National Science Foundation Grant IIS-9978567 and by a Sloan Fellowship.

[*]This work was done while the author was at the University of Washington.

In a data integration system, users pose queries in terms of a virtual *mediated schema*. The mediated schema is virtual in the sense that the data is not actually stored in this schema, but rather in the data sources in their own schemas. Hence, in order to answer queries, the system includes a set of *source descriptions* that provide the semantic mapping between the relations in the mediated schema and the relations in the source schema. One of the common approaches to specifying source descriptions (known as *local-as-view*) is to describe data sources as containing answers to views over the mediated schema [32; 17; 20; 30].

The semantics of a query in such a setting can be formalized in terms of *certain answers* [1]. Intuitively, a tuple $\bar{t}$ is a certain answer to a query Q over the mediated schema with respect to a set of source instances if $\bar{t}$ is an answer in any database D over the mediated schema that is consistent with the source instances. In some cases, the set of certain answers can be obtained by algorithms for rewriting queries using views [40; 31; 38; 10], while in others, specialized algorithms are necessary [1; 23].

In this paper we consider the problem of query containment in data integration systems. In the relational model, a query Q_1 is said to be contained in the query Q_2 if Q_1 produces a subset of the answers of Q_2 for any given database. Query containment has been considered for the purposes of query optimization [9; 35; 3], detecting independence of queries from database updates [33], rewriting queries using views [38; 31], maintenance of integrity constraints [24], and semantic data caching [14; 27; 12; 2].

In the context of data integration, we need to refine our notion of containment to consider the set of available sources. Intuitively, we will say that a query Q_1 is contained in Q_2 relative to the sources if, for any instance of the data sources, the certain answers of Q_1 are a subset of the certain answers of Q_2. We formalize this notion as query containment *relative to views*.

EXAMPLE 1. *Consider a mediated schema that includes two relations.*

$$CarDesc(CarNo, Model, Color, Year)$$
$$Review(Model, Review, Rating)$$

$CarDesc$ describes cars for sale, including their number, model, color and year of manufacture. Review provides the review and numerical rating from 1 to 10 that have been given to particular car models.

The following are three possible data sources. The first source, $RedCars$, provides listings of red cars, while the second source, *AntiqueCars, provides listings of cars manufactured before 1970. The third source provides reviews, but only for models given the top rating (10).*

$$\begin{aligned}
&\mathit{RedCars}(\mathit{CarNo}, \mathit{Model}, \mathit{Year}) \subseteq\\
&\qquad \mathit{CarDesc}(\mathit{CarNo}, \mathit{Model}, \mathit{red}, \mathit{Year}).\\
&\mathit{AntiqueCars}(\mathit{CarNo}, \mathit{Model}, \mathit{Year}) \subseteq\\
&\qquad \mathit{CarDesc}(\mathit{CarNo}, \mathit{Model}, \mathit{Color}, \mathit{Year}),\\
&\qquad \mathit{Year} < 1970.\\
&\mathit{CarAndDriver}(\mathit{Model}, \mathit{Review}) \subseteq\\
&\qquad \mathit{Review}(\mathit{Model}, \mathit{Review}, 10).
\end{aligned}$$

Consider the following three queries:

$$\begin{aligned}
&Q_1 : q_1(\mathit{CarNo}, \mathit{Review}) \text{ :-}\\
&\qquad \mathit{CarDesc}(\mathit{CarNo}, \mathit{Model}, C, Y),\\
&\qquad \mathit{Review}(\mathit{Model}, \mathit{Review}, \mathit{Rating}).\\
&Q_2 : q_2(\mathit{CarNo}, \mathit{Review}) \text{ :-}\\
&\qquad \mathit{CarDesc}(\mathit{CarNo}, \mathit{Model}, C, Y),\\
&\qquad \mathit{Review}(\mathit{Model}, \mathit{Review}, 10).\\
&Q_3 : q_3(\mathit{CarNo}, \mathit{Review}) \text{ :-}\\
&\qquad \mathit{CarDesc}(\mathit{CarNo}, \mathit{Model}, C, Y),\\
&\qquad \mathit{Review}(\mathit{Model}, \mathit{Review}, 10), Y < 1990.
\end{aligned}$$

Query Q_1 asks for car reviews, while Q_2 asks for car reviews for the finest models. Query Q_3 asks for reviews of the finest models built before 1990. In the traditional context, the query Q_2 is contained in query Q_1 because Q_2 applies a stronger condition ($Rating = 10$) than Q_1, but Q_1 is not contained in Q_2. Likewise, Q_3 is contained in Q_2, but not vice versa. However, because reviews are only available for top-rated cars, Q_1 is contained in Q_2 relative to the sources, and in fact the two queries return the same certain answers. On the other hand, Q_1 is not contained in Q_3 relative to the sources, because it is possible to retrieve reviews of red cars made after 1990. If the RedCars source were not available, then Q_1 would be contained in Q_3 relative to the available sources. □

As the above example illustrates, aside from the traditional uses of query containment, an additional use in the data integration framework is to familiarize a user with the coverage and limitations of a large set of available data sources. For example, the system can tell the user whether the answers to two queries Q_1 and Q_2 are the same because the queries are equivalent, or because they are equivalent for the current available sources.

We establish several fundamental complexity results about query containment relative to views. We begin with the case in which queries and views are conjunctive and do not include comparison predicates. In this case, the set of certain answers to each query can be obtained by a non-recursive datalog program [1]. Therefore, determining query containment of the datalog programs created for the two queries also determines containment of the queries relative to the views. The complexity of producing those datalog programs and checking their containment is an upper bound on our problem. We also provide a tight lower bound for this case, and matching bounds for the case of positive queries.

Next, we consider the case where there are limitations on access patterns to the underlying sources. Here, it follows from [15] that even for conjunctive queries and views, the set of certain answers can only be obtained by a *recursive* datalog program over the views. Hence, a solution to the relative containment problem based on comparing the datalog programs created for the two queries will not work, since containment of datalog programs is undecidable in general [36]. However, we show that relative containment *is* decidable in this case.

Finally, we consider cases in which the queries and the views contain comparison predicates. We provide tight complexity bounds on relative containment for two interesting variants of the problem.

In this paper we consider data integration systems where the source descriptions are provided using the local-as-view approach. In a second approach, known as global-as-view, the mediated schema is described as a set of views over the source relations. We note that using that approach, algorithms and complexity results for relative containment are straightforward corollaries of traditional query containment results.

2. PRELIMINARIES

We begin by defining the terms used throughout the paper.

2.1 Queries and Views

In our discussion we consider queries and views in datalog or some subset thereof. A datalog program is a set of datalog rules. A datalog rule has the form:

$$q(\bar{X}) :- r_1(\bar{X}_1),\ \ldots,\ r_n(\bar{X}_n)$$

where q and $r_1, \ldots, r_n$ are *predicate* (also called *relation*) names. The atom $q(\bar{X})$ is called the *head* of the rule, and the atoms $r_1(\bar{X}_1), \ldots, r_n(\bar{X}_n)$ are the *subgoals* in the body of the rule. The tuples $\bar{X}, \bar{X}_1, \ldots, \bar{X}_n$ contain either variables or constants. We require that every rule be *safe* — every variable that appears in the head must also appear in the body. We say that the head of a query is *empty* if $\bar{X} = \{\}$.

The predicate names in a datalog rule range over the *extensional database* (EDB) predicates, which are the relations stored in the database, and the *intensional database* (IDB) predicates (like q above), which are relations defined by datalog rules. EDB predicates may not appear in the head of a datalog rule.

A *conjunctive query* is a datalog program consisting of a single rule, whose body therefore contains only EDB predicates. A nonrecursive datalog program (also called a *positive query*) is a set of datalog rules such that there exists an ordering $R_1, \ldots, R_m$ of the rules so that the predicate name in the head of R_i does not occur in the body of rule R_j whenever $j \leq i$. Such datalog programs can always be *unfolded* into a finite union of conjunctive queries.

In Section 5 we consider queries and views with comparison predicates $\neq, <, >, \leq$, and $\geq$, interpreted over a dense domain. In this case, we require that for each datalog rule, if a variable X appears in a subgoal with a comparison predicate, then X must also appear in an ordinary subgoal in the body of the rule.

A query Q_1 is said to be *contained* in a query Q_2, denoted by $Q_1 \sqsubseteq Q_2$, if the set of answers of $Q_1(D)$ is a subset of those of $Q_2(D)$ for *any* database D. Q_1 and Q_2 are *equivalent* if each is contained in the other.

2.2 Mediated Schemas and Source Descriptions

Users of a data integration system pose queries in terms of a mediated schema. The data, however, is stored in the data

sources using their local schemas. Hence, in order to translate a user query posed over the mediated schema into a query on the sources, the system contains a set of source descriptions that provides the semantic mapping between the relations in the mediated schema and those in the sources.
In our discussion, we consider source descriptions of the form

$$V(\bar{X}) \subseteq Q(\bar{X})$$

where Q is a query (over the mediated schema) and V is one of the relations exported by a data source. The meaning of such a description is that the relation V in the source contains tuples that are answers to the query Q over the mediated schema. In practice we use a slight abuse of notation, where the atom $V(\bar{X})$ also stands for the head of the query Q.
We distinguish between incomplete and complete sources. Incomplete sources are not assumed to include all the answers to Q, but just some subset of the answers. Complete sources are assumed to contain all the answers to Q, and are denoted by descriptions in which the $\subseteq$ is replaced by $\equiv$. We focus here on incomplete sources and comment on complete ones in Section 6. ([1] refers to the case of incomplete/complete sources as the open-world/closed-world assumption).

2.3 Certain Answers

The answer to a query in a data integration system is formalized by the notion of *certain answers* [1]:

DEFINITION 2.1. (**Certain answers**) *Given a query Q, a set of (possibly incomplete) sources $\mathcal{V} = V_1, \ldots, V_n$ described as views, and instances $I = v_1, \ldots, v_n$ for the sources, the tuple $\bar{t}$ is a* certain answer *to Q w.r.t. I if $\bar{t}$ is in $Q(D)$ for any database D such that $I \subseteq \mathcal{V}(D)$. We denote the set of certain answers of Q given I by $certain(Q, I)$.* $\square$

The complexity of finding certain answers is discussed in [1; 23]. When the query is in datalog and does not contain comparison predicates, and the views are conjunctive, the set of certain answers can be obtained by a *query plan*, a datalog program whose EDB relations are the source relations.
Given a query plan P for Q using $\mathcal{V}$, the *expansion* P^{exp} of P is obtained from P by replacing all source relations in P with the bodies of the corresponding views in $\mathcal{V}$, and using fresh variables for existential variables in the views [15].
The query plan that produces all certain answers is called the *maximally-contained* plan, and is formally defined as follows [15]:

DEFINITION 2.2. (**Maximally-contained plan**) *A query plan P is maximally-contained in a query Q using views $\mathcal{V}$ if $P^{exp} \sqsubseteq Q$ and for every query plan P' such that $P'^{exp} \sqsubseteq Q$, $P' \sqsubseteq P$.* $\square$

The maximally-contained query plan is obtained by an algorithm for rewriting queries using views [31; 40; 38; 10; 16]. In our discussion, we rely on the properties of one such algorithm, the *inverse rules* algorithm [15]. Informally, the algorithm simply "inverts" each view definition, producing several datalog rules, one per non-comparison subgoal in the body of the view. Existential variables in the body of a view become function terms in the head of an inverted rule, ensuring that each inverted rule is still safe. The maximally-contained query plan is then the union of the original query and the set of inverted view definitions.

EXAMPLE 2. *The maximally-contained query plan constructed by the inverse rules algorithm for query Q_1 from Example 1, given the three sources listed there, is the following plan P_1:*

$p_1(CarNo, Review)$:-
  $CarDesc(CarNo, Model, C, Y)$,
  $Review(Model, Review, Rating)$.
$CarDesc(CarNo, Model, red, Year)$:-
  $RedCars(CarNo, Model, Year)$.
$CarDesc(CarNo, Model, f(CarNo, Model, Year), Year)$:-
  $AntiqueCars(CarNo, Model, Year)$.
$Review(Model, Review, 10)$:-
  $CarAndDriver(Model, Review)$.

Since Q_2 from Example 1 is equivalent to Q_1 relative to the sources, the above query plan is also maximally-contained for Q_2.

The maximally-contained query plan of a positive query is positive, and the maximally-contained query plan of a recursive query is recursive [15]. As mentioned above and shown in Example 2, the query plan may contain function terms; [15] shows how to remove these function terms. This procedure results in a positive query plan if the original query plan was positive.

EXAMPLE 3. *Removing function terms from the query plan in Example 2 and unfolding the resulting plan to a union of conjunctive queries results in the following equivalent plan P_1':*

$p_1'(CarNo, Review)$:-
  $RedCars(CarNo, Model, Year)$,
  $CarAndDriver(Model, Review)$.
$p_1'(CarNo, Review)$:-
  $AntiqueCars(CarNo, Model, Year)$,
  $CarAndDriver(Model, Review)$.

As shown in [1], when the query contains comparison predicates or when the sources are assumed to be complete, the problem of finding certain answers becomes co-NP-hard in the size of the source instances, even when the sources and the query are conjunctive. Since datalog programs have polynomial data complexity, a query plan in datalog that produces all certain answers cannot always be found in these cases. Restricted cases where the query can contain comparison predicates but the certain answers can be found in polynomial time are given in [21].

2.4 Relative Containment

In the context of data integration we need to modify the standard relational notion of query containment. Although it is the case that if Q_1 is contained in Q_2, then Q_1 is contained in Q_2 relative to the available data sources, the converse does not hold. In particular, Q_1 may always produce a subset of the answers produced by Q_2 given the available data sources, even if Q_1 is not contained in Q_2. As shown in Example 1, two queries that are not equivalent according to the traditional definition may be equivalent relative to a set of sources. We formalize the notion of containment in a data integration system as follows:

DEFINITION 2.3. (**Relative containment**) *Given a set of views $\mathcal{V} = V_1, \ldots, V_n$ and queries Q_1 and Q_2, we say that Q_1 is contained in Q_2 relative to $\mathcal{V}$, denoted by $Q_1 \sqsubseteq_{\mathcal{V}}$*

Q_2, if for any instance I of the views $\mathcal{V}$, $certain(Q_1, I) \subseteq certain(Q_2, I)$. □

2.5 Combined Complexity

The standard complexity measure for query containment in the relational setting is *query complexity*, which measures the complexity of containment as the queries vary. However, for relative containment, both the sizes of the queries and the views can be important factors on the problem's complexity. Therefore, throughout the paper, our complexity results use the *combined query and view complexity*, which measures the complexity of relative containment as both the queries and views vary.

3. COMPLEXITY OF RELATIVE CONTAINMENT

In this section, we provide tight bounds on the complexity of relative containment when the view definitions are conjunctive.

3.1 Upper Bounds

Let Q_1 and Q_2 be datalog programs and $\mathcal{V}$ be a set of conjunctive views. As described in the previous section, algorithms exist for constructing a query plan P_1 (P_2) that produces all certain answers to Q_1 (Q_2). Therefore, a simple way to decide whether $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$ is to construct P_1 and P_2 and check whether $P_1 \sqsubseteq P_2$. This idea provides the following upper bounds on the complexity of relative containment:

THEOREM 3.1. *Given two nonrecursive datalog programs Q_1 and Q_2, and a set $\mathcal{V}$ of conjunctive views, determining whether $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$ is in Π_2^P.*

Proof sketch: Since Q_1 (Q_2) is a nonrecursive datalog program, so is P_1 (P_2), the associated maximally-contained query plan. Therefore, P_1 and P_2 are each equivalent to a finite union of conjunctive queries. In particular, it is easy to see by the definition of maximal containment that P_1 is equivalent to the union of every conjunctive query plan CQ_1 such that $CQ_1^{exp} \sqsubseteq Q_1$, and similarly for P_2. Although there may be an infinite number of such conjunctive query plans in general, it is shown in [31] that it suffices to consider only conjunctive query plans of size no more than the size of the original query.

Therefore, suppose that Q_1 has a total of n subgoals in all of its rules, and Q_2 has a total of m subgoals in all of its rules. To decide whether $P_1 \sqsubseteq P_2$, we check whether for every conjunctive query plan CQ_1 with at most n subgoals such that $CQ_1^{exp} \sqsubseteq Q_1$, there exists a conjunctive query plan CQ_2 with at most m subgoals such that $CQ_2^{exp} \sqsubseteq Q_2$ and $CQ_1 \sqsubseteq CQ_2$. Since deciding containment of a conjunctive query in a positive query is in NP [3], the time complexity of this algorithm for each CQ_1 is in NP, so the total time is in Π_2^P. □

THEOREM 3.2. *Given two datalog programs Q_1 and Q_2, where at most one of Q_1 and Q_2 is recursive, and a set $\mathcal{V}$ of conjunctive views, it is decidable whether $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$.*

Proof: As mentioned above, we can construct datalog query plans P_1 and P_2 such that $Q_1 \sqsubseteq_{\mathcal{V}} Q_2 \iff P_1 \sqsubseteq P_2$. Since at most one of Q_1 and Q_2 is recursive, it is also the case that at most one of P_1 and P_2 is recursive. Therefore, it is decidable whether $P_1 \sqsubseteq P_2$ [11]. □

3.2 Lower Bounds

In this section, we show that the upper bound of Theorem 3.1 is tight, even when both Q_1 and Q_2 are conjunctive queries:

THEOREM 3.3. *Given two conjunctive queries Q_1 and Q_2 and a set $\mathcal{V}$ of conjunctive views, determining whether $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$ is Π_2^P-hard.*

Proof sketch: We reduce the $\forall\exists$-CNF problem, known to be Π_2^P-complete [37], to our problem. The $\forall\exists$-CNF problem is defined as follows: Given a 3-CNF formula F with variables $\overline{x}$ and $\overline{y}$, is it the case that for each truth assignment to $\overline{y}$, there exists a truth assignment to $\overline{x}$ that satisfies F?

We are given a 3-CNF formula F, with variables

$$\overline{z} = \{x_1, \dots, x_n\} \cup \{y_1, \dots, y_m\}$$

and clauses $\overline{c} = \{c_1, \dots, c_p\}$. Clause c_i contains the three variables (either positive or negated) $z_{i,1}, z_{i,2}$, and $z_{i,3}$.

We begin by building Q'_1 and Q'_2, in a similar fashion to the reduction proof used in [3] to show NP-hardness of conjunctive query containment. We use the EDB predicates $r_1, \dots, r_p$ of arity 3, each predicate representing a different clause in F. Query Q'_1 simply records which variables are in each clause. It is defined as follows:

$$q'_1() \text{ :- } r_1(z_{1,1}, z_{1,2}, z_{1,3}), \dots, r_p(z_{p,1}, z_{p,2}, z_{p,3}).$$

Query Q'_2 records all seven satisfying assignments for each clause in F. The head of Q'_2 is empty. For each clause c_i in F, for each truth assignment $\{a_{i,1}, a_{i,2}, a_{i,3}\}$ to $\{z_{i,1}, z_{i,2}, z_{i,3}\}$ that satisfies c_i, the body of Q'_2 contains the subgoal

$$r_i(a_{i,1}, a_{i,2}, a_{i,3}).$$

For example, consider the formula $(x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$. We build the following two queries:

$$\begin{array}{ll} q'_1() \text{ :- } & r_1(x_1, x_2, y_1), r_2(x_1, x_2, y_2). \\ q'_2() \text{ :- } & r_1(1,1,1), r_1(1,1,0), r_1(1,0,1), r_1(1,0,0), \\ & r_1(0,1,1), r_1(0,1,0), r_1(0,0,1), r_2(0,0,0), \\ & r_2(0,0,1), r_2(0,1,0), r_2(0,1,1), r_2(1,0,0), \\ & r_2(1,0,1), r_2(1,1,0). \end{array}$$

Similar to the argument in [3], it can be shown that this is a valid reduction from the CNF-satisfiability problem to the conjunctive query containment problem: F is satisfiable if and only if $Q'_2 \sqsubseteq Q'_1$. In particular, any satisfying truth assignment for F is also a valid containment mapping from Q'_1 to Q'_2, and vice versa.

We now build the queries Q_1 and Q_2 by adding subgoals to Q'_1 and Q'_2, respectively. To create Q_1, we add to Q'_1 the new subgoals $e_1(y_1), \dots, e_m(y_m)$, where e is a new EDB predicate. To create Q_2 we add to Q'_2 the new subgoals $e_1(u_1), \dots, e_m(u_m)$, where each u_i is a fresh variable.

Now we create the views. First we have p views that simply mirror each of the r_i predicates: $v_i(z_1, z_2, z_3) \text{ :- } r_i(z_1, z_2, z_3)$. Finally, for each variable y_i, we have two views that represent that variable being assigned the truth value zero and one respectively: $w_{i,0}() \text{ :- } e_i(0)$ and $w_{i,1}() \text{ :- } e_i(1)$.

For our example formula, $(x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_2)$, we build the following queries and views:

$$\begin{array}{ll}
q_1() \text{ :- } & r_1(x_1,x_2,y_1), r_2(x_1,x_2,y_2), e_1(y_1), e_2(y_2). \\
q_2() \text{ :- } & r_1(1,1,1), r_1(1,1,0), r_1(1,0,1), r_1(1,0,0), \\
 & r_1(0,1,1), r_1(0,1,0), r_1(0,0,1), r_2(0,0,0), \\
 & r_2(0,0,1), r_2(0,1,0), r_2(0,1,1), r_2(1,0,0), \\
 & r_2(1,0,1), r_2(1,1,0), e_1(u_1), e_2(u_2). \\
v_1(z_1,z_2,z_3) \text{ :- } & r_1(z_1,z_2,z_3). \\
v_2(z_1,z_2,z_3) \text{ :- } & r_2(z_1,z_2,z_3). \\
w_{1,0}() \text{ :- } & e_1(0). \\
w_{1,1}() \text{ :- } & e_1(1). \\
w_{2,0}() \text{ :- } & e_2(0). \\
w_{2,1}() \text{ :- } & e_2(1).
\end{array}$$

We now show that F is $\forall\exists$-satisfiable if and only if $Q_2 \sqsubseteq_{\mathcal{V}} Q_1$. Let P_1 (P_2) be the query plan that produces all certain answers to Q_1 (Q_2), constructed by the inverse rules algorithm. In particular, P_1 consists of the union of the rule defining Q_1 and the inverted view definitions, and similarly for P_2. In our example, the inverted view definitions are as follows:

$$\begin{array}{ll}
r_1(z_1,z_2,z_3) \text{ :- } & v_1(z_1,z_2,z_3). \\
r_2(z_1,z_2,z_3) \text{ :- } & v_2(z_1,z_2,z_3). \\
e_1(0) \text{ :- } & w_{1,0}(). \\
e_1(1) \text{ :- } & w_{1,1}(). \\
e_2(0) \text{ :- } & w_{2,0}(). \\
e_2(1) \text{ :- } & w_{2,1}().
\end{array}$$

We know that $Q_2 \sqsubseteq_{\mathcal{V}} Q_1 \iff P_2 \sqsubseteq P_1$. Since the queries are conjunctive, P_1 and P_2 are nonrecursive datalog programs. Further, by construction of the queries and views, P_1 and P_2 do not contain function terms. Therefore, we can unfold P_1 (P_2) into an equivalent union of conjunctive queries P_1* (P_2*). Therefore, $P_2 \sqsubseteq P_1 \iff P_2* \sqsubseteq P_1*$.
It is easy to see that P_1* and P_2* are each a union of 2^m conjunctive queries, one per truth assignment to $\overline{y}$. In particular, the conjunctive query CQ_1^A in P_1*, corresponding to truth assignment $A = \{a_1, \ldots, a_m\}$ to $\overline{y}$, is identical to Q_1 with the following modifications: Each predicate name r_i is replaced by the predicate name v_i, each subgoal $e_i(y_i)$ is replaced by $w_{i,a_i}()$, and each occurrence of variable y_i is replaced by the constant a_i. Similarly, the conjunctive query CQ_2^A in P_2*, corresponding to truth assignment $A = \{a_1, \ldots, a_m\}$ to $\overline{y}$, is identical to Q_2 with the following modifications: Each predicate name r_i is replaced by the predicate name v_i, and each subgoal $e_i(w_i)$ is replaced by $w_{i,a_i}()$.
For example, with the truth assignment $A = \{1, 0\}$ for $\{y_1, y_2\}$, the associated conjunctive queries in P_1* and P_2* in our example are as follows:

$$\begin{array}{ll}
cq_1^A() \text{ :- } & v_1(x_1,x_2,1), v_2(x_1,x_2,0), w_{1,1}(), w_{2,0}(). \\
cq_2^A() \text{ :- } & v_1(1,1,1), v_1(1,1,0), v_1(1,0,1), v_1(1,0,0), \\
 & v_1(0,1,1), v_1(0,1,0), v_1(0,0,1), v_2(0,0,0), \\
 & v_2(0,0,1), v_2(0,1,0), v_2(0,1,1), v_2(1,0,0), \\
 & v_2(1,0,1), v_2(1,1,0), w_{1,1}(), w_{2,0}().
\end{array}$$

By [35], $P_2* \sqsubseteq P_1*$ if and only if each conjunctive query in P_2* is contained in some conjunctive query in P_1*. Consider the conjunctive query CQ_2^A in P_2*, corresponding to some truth assignment $A = \{a_1, \ldots, a_m\}$ to $\overline{y}$. We know that CQ_2^A has precisely the following w subgoals in its body: $w_{1,a_1}(), \ldots, w_{m,a_m}()$. Therefore, a containing query from P_1* must also have precisely those w subgoals in its body, so the containing query can only possibly be CQ_1^A, the conjunctive query in P_1* corresponding to the same truth assignment A to $\overline{y}$. Therefore, we have that $P_2* \sqsubseteq P_1*$ if and only if for each assignment A to $\overline{y}$, $CQ_2^A \sqsubseteq CQ_1^A$.
Let CQ_2^{A-} be CQ_2^A with all of the w subgoals removed from its body, and similarly for CQ_1^{A-}. Since for each assignment A to $\overline{y}$, CQ_2^A and CQ_1^A have the same w subgoals in their bodies, we have $CQ_2^A \sqsubseteq CQ_1^A \iff CQ_2^{A-} \sqsubseteq CQ_1^{A-}$. Note that CQ_2^{A-} is equivalent to Q_2' above, and CQ_1^{A-} is equivalent to Q_1', but with the $\overline{y}$ variables hard-coded to the truth assignment A. Thus, from the correctness of the CNF-satisfiability reduction to conjunctive query containment, we know that for each assignment A to $\overline{y}$, $CQ_2^{A-} \sqsubseteq CQ_1^{A-}$ if and only if there exists a truth assignment A' to $\overline{x}$ such that $A' \cup A$ satisfies F. By definition, this is true if and only if F is $\forall\exists$-satisfiable. $\Box$

The combination of Theorems 3.1 and 3.3 implies that relative containment for conjunctive queries and views is Π_2^P-complete. In contrast, ordinary conjunctive query containment is NP-complete [3]. The theorems also imply Π_2^P-completeness for relative containment when both queries are positive, matching the complexity of ordinary query containment in this case [35].

4. BINDING PATTERN LIMITATIONS

In this section we consider the common case of data sources that have access pattern limitations. For example, when accessing Amazon.com, one cannot ask for all books and their prices. Instead, one obtains the price of a book only if the ISBN is given as input. The access limitations to such sources are modeled by adornments in the source descriptions. An adornment is a string of b's and f's whose length is the number of variables in the predicate defined by the source description. A b stands for "bound," which means that the value must be provided to the source, and an f stands for "free," which means that the value need not be provided. For example, the following adornment on $RedCars$ indicates that the model of the car needs to be provided in order to obtain the cars for sale of that model.

$$\begin{array}{l}
RedCars^{fbf}(CarNo, Model, Year) \subseteq \\
\qquad CarDescription(CarNo, Model, red, Year).
\end{array}$$

We concentrate here on the case where each source has a single adornment. Sources with multiple possible access patterns can be modelled by a set of adornments, and it is straightforward to generalize our results for that case.

4.1 Problem Definition

Before we can discuss relative containment in this context, we need to extend the notion of certain answers to take into consideration the access restrictions on sources. In what follows we define the revised notion of certain answers for the case in which they can be obtained by a datalog program. Although a revision for the more general case is possible, we omit it here. We first define executable datalog programs, which are datalog programs that obey the binding pattern restrictions.

Definition 4.1. (**Executability**) *A datalog rule is* executable *if for each EDB subgoal $r(\bar{X})$ in its body, for each position i of r's adornment that is a b, either the ith parameter of $r(\bar{X})$ is a constant or it is a variable that also* *appears to the left of this occurrence. A datalog program is executable if each of its rules is executable.* □

Access pattern limitations have the effect of possibly limiting the set of constants that can be obtained from the data sources. However, a query plan can "cheat" by introducing new constants. For example, consider the following query, which asks for the number and year of all red cars:

$$Q : q(CarNo, Year) \text{ :- } CarDescription(CarNo, Model, red, Year).$$

Given only the revised *RedCars* source above, we cannot find any answers to Q because the *RedCars* source requires that the car model be provided. However, it is possible to construct an executable query plan that *does* find an answer to Q, by inventing a particular constant for the car model:

$$p(CarNo, Year) \text{ :- } RedCars(CarNo, corolla, red, Year).$$

If the *RedCars* source contains a listing for a red Corolla, this query plan will return an answer to Q.
To eliminate such spurious query plans from consideration, the following definition considers only plans that introduce no new constants:

DEFINITION 4.2. *(***Sound query plan***) Given a query Q, views $\mathcal{V}$, and a set B of one binding pattern adornment for each view predicate, a query plan P is* sound *relative to Q, $\mathcal{V}$, and B if P is executable, the constants in P are a subset of those in $Q \cup V$, and $P^{exp} \sqsubseteq Q$.* □

Intuitively, a sound query plan is one that obeys the access restrictions on sources and produces only answers that the original query would also produce. This is precisely the notion needed to strengthen the definition of certain answers. In particular, we only want to consider certain answers that can be produced by a sound query plan:

DEFINITION 4.3. *(***Reachable certain answers***) Given a query Q, a set of sources $\mathcal{V} = V_1, \ldots, V_n$, a set B of one binding pattern adornment for each view predicate, and instances $I = v_1, \ldots, v_n$ for the sources, the tuple $\bar{t}$ is a* reachable certain answer *to Q w.r.t. I if $\bar{t}$ is a certain answer to Q w.r.t. I and there exists a sound query plan P relative to $Q, \mathcal{V}$, and B such that $\bar{t} \in P(I)$.* □

The problem of answering queries using views with binding pattern limitations is considered in [34; 29; 15; 30]. The results of [15] imply that, when the query is in datalog and the views are conjunctive, the set of reachable certain answers can always be obtained by a *recursive* datalog program. Further, recursion is necessary in general for finding all such answers, even when the query is conjunctive. This datalog program is maximally-contained in the following sense:

DEFINITION 4.4. *(***Maximal containment with binding patterns***) A query plan P is maximally-contained in a query Q using views $\mathcal{V}$ and a set B of one binding pattern adornment for each view predicate if P is sound relative to $Q, \mathcal{V}$, and B, and for every query plan P' that is sound relative to $Q, \mathcal{V}$, and B, $P' \sqsubseteq P$.* □

Finally, the relative containment problem in the presence of binding pattern restrictions on sources is defined as follows:

DEFINITION 4.5. *(***Relative containment***) Given a set of views $\mathcal{V} = V_1, \ldots, V_n$, a set B of one binding pattern adornment for each view predicate, and queries Q_1 and Q_2 such that $Q_1 \cup \mathcal{V}$ has a subset of the constants in $Q_2 \cup \mathcal{V}$, we say that Q_1 is contained in Q_2 relative to $\mathcal{V}$ and the binding patterns, denoted by $Q_1 \sqsubseteq_{\mathcal{V},B} Q_2$, if for any instance I of the views, the reachable certain answers of Q_1 are a subset of the reachable certain answers of Q_2.* □

4.2 Decidability

As in Section 3, we know that $Q_1 \sqsubseteq_{V,B} Q_2 \iff P_1 \sqsubseteq P_2$, where P_1 (P_2) is the maximally-contained query plan of Q_1 (Q_2). As mentioned above, P_1 and P_2 are recursive in general, even if Q_1 and Q_2 are conjunctive queries. Therefore, checking containment of P_1 and P_2 does not provide a useful upper bound on the complexity of relative containment, because containment of arbitrary recursive datalog programs is undecidable [36].
However, the following theorem shows that, surprisingly, to check relative containment it suffices to consider the effect of the views and the binding patterns only on the possibly contained query:

THEOREM 4.1. *Let $\mathcal{V}$ be a set of conjunctive view definitions, let B be a set of one binding pattern adornment for each view predicate, and let Q_1 and Q_2 be datalog queries such that $Q_1 \cup \mathcal{V}$ has a subset of the constants in $Q_2 \cup \mathcal{V}$. Let P_1 and P_2 be the maximally-contained query plans of Q_1 and Q_2 using $\mathcal{V}$ and B. Then, $P_1 \sqsubseteq P_2 \iff P_1^{exp} \sqsubseteq Q_2$.*

Proof: For the "⇒" direction, $P_1 \sqsubseteq P_2 \iff \forall I. P_1(I) \subseteq P_2(I) \Rightarrow \forall D. P_1(\mathcal{V}(D)) \subseteq P_2(\mathcal{V}(D)) \iff P_1^{exp} \sqsubseteq P_2^{exp}$. Since P_2 is the maximally-contained query plan for Q_2 using $\mathcal{V}$ and B, $P_2^{exp} \sqsubseteq Q_2$, so transitively we have $P_1^{exp} \sqsubseteq Q_2$.
For the "⇐" direction, suppose $P_1^{exp} \sqsubseteq Q_2$. Since P_1 is the maximally-contained query plan for Q_1 using $\mathcal{V}$ and B, P_1 is executable and has a subset of the constants in $Q_1 \cup \mathcal{V}$. We are given that $Q_1 \cup \mathcal{V}$ has a subset of the constants in $Q_2 \cup \mathcal{V}$, so transitively P_1 does as well. Therefore, P_1 is sound relative to Q_2, $\mathcal{V}$, and B. Since P_2 is the maximally-contained query plan for Q_2 and $\mathcal{V}$, by the definition of maximal containment we have $P_1 \sqsubseteq P_2$. □
As the above theorem shows, when Q_2 is nonrecursive, we can reduce relative containment of Q_1 and Q_2 to the ordinary query containment of a recursive datalog program in a non-recursive one. Hence, the decidability results of [11] in this case entail the following theorem:

THEOREM 4.2. *Given a (potentially recursive) datalog program Q_1, a non-recursive datalog program Q_2, a set $\mathcal{V}$ of conjunctive views, and a set B of one binding pattern adornment for each view predicate, determining whether $Q_1 \sqsubseteq_{\mathcal{V},B} Q_2$ is decidable.*

Note that an analogous version of Theorem 4.1 can be proven for the ordinary relative containment problem without binding patterns discussed in Section 3. However, we did not require such a result to obtain the complexity bounds in that case.

5. COMPARISON PREDICATES

In this section we study the complexity of relative containment when queries and views may contain comparison predicates of the form $\neq, <, >, \leq$, and $\geq$.

In general, finding all certain answers to a query that contains comparison predicates is co-NP-hard in the size of the view instances [1; 19], so datalog query plans do not always exist. However, there are certain cases where it is known that maximally-contained datalog query plans will exist [21]. One such case is when all comparison atoms are *semi-interval.* A comparison atom is a semi-interval constraint if it has the form $x\theta c$, where x is a variable, c is a constant, and θ is either $<$ or $\leq$ (alternatively, θ is either $>$ or $\geq$). Therefore, we can show the following:

THEOREM 5.1. *Given positive queries Q_1 and Q_2 and conjunctive views $\mathcal{V}$, where all queries and views may contain semi-interval constraints, the problem of determining whether $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$ is in Π_2^P.*

The proof of the theorem follows from a parallel argument to that of Theorem 3.1. In particular, the proof relies on the fact that containment of conjunctive queries with semi-interval constraints is still in NP [28], and that when building the maximally-contained plan for a positive query with semi-interval constraints, even in the presence of semi-interval constraints in the views, we still need only consider conjunctive query plans whose length (in terms of the number of non-comparison subgoals) is at most that of the query. Once the non-comparison subgoals are chosen for a candidate conjunctive query plan, it is straightforward to pick the appropriate semi-interval constraints to add (or to show that no appropriate constraints exist, in which case the candidate is not part of the maximally-contained query plan for the query).

EXAMPLE 4. *The maximally-contained query plan P_3 for query Q_3 in Example 1 is as follows:*

$p_3(CarNo, Review)$:-
  $RedCars(CarNo, Model, Year)$,
  $CarAndDriver(Model, Review), Year < 1990$.
$p_3(CarNo, Review)$:-
  $AntiqueCars(CarNo, Model, Year)$,
  $CarAndDriver(Model, Review)$.

Because P_3 does not contain plan P_1' from Example 3, which is the maximally-contained query plan for Q_1, we know that Q_3 does not contain Q_1 relative to the views.

There is another case where we can provide a tight bound on the complexity of relative containment with comparison predicates. It is especially interesting because it is a case where finding the certain answers of Q_2 is co-NP-hard, and hence a maximally-contained datalog query plan for Q_2 does not exist.

First we prove a theorem similar in spirit to Theorem 4.1. It relies on the fact that, given a positive query Q_1 and conjunctive views $\mathcal{V}$, we can construct the maximally-contained query plan P_1 for Q_1 even if the views may contain arbitrary comparison predicates. In particular, since Q_1 does not contain any comparison predicates, we can use standard algorithms for rewriting queries using views to construct P_1.

THEOREM 5.2. *Given positive queries Q_1 and Q_2 and conjunctive views $\mathcal{V}$, where Q_2 and $\mathcal{V}$ may contain arbitrary comparison predicates, let P_1 be the maximally-contained query plan for Q_1. Then $Q_1 \sqsubseteq_{\mathcal{V}} Q_2 \iff P_1^{exp} \sqsubseteq Q_2$.*

Proof: For the "$\Rightarrow$" direction,

$$\begin{array}{ll}
Q_1 \sqsubseteq_{\mathcal{V}} Q_2 & \Rightarrow \\
\forall I.certain(Q_1, I) \subseteq certain(Q_2, I) & \Rightarrow \\
\forall I.P_1(I) \subseteq certain(Q_2, I) & \Rightarrow \\
\forall D.P_1(\mathcal{V}(D)) \subseteq certain(Q_2, \mathcal{V}(D)) & \Rightarrow \\
\forall D.P_1(\mathcal{V}(D)) \subseteq Q_2(D) & \Rightarrow \\
P_1^{exp} \sqsubseteq Q_2 &
\end{array}$$

For the "$\Leftarrow$" direction, we prove the contrapositive. Suppose $Q_1 \not\sqsubseteq_{\mathcal{V}} Q_2$. Then there exists an instance I of the views and a tuple t such that $t \in certain(Q_1, I)$ but $t \notin certain(Q_2, I)$. Since $t \notin certain(Q_2, I)$, there exists some database D' such that $I \subseteq \mathcal{V}(D')$ and $t \notin Q_2(D')$. On the other hand, since $t \in certain(Q_1, I)$, we know that $t \in P_1(I)$. Further, by the monotonicity of P_1, $t \in P_1(I')$ for any I' such that $I \subseteq I'$. So in particular, $t \in P_1(\mathcal{V}(D'))$, so $t \in P_1^{exp}(D')$. Since $t \in P_1^{exp}(D')$ but $t \notin Q_2(D')$, we have that $P_1^{exp} \not\sqsubseteq Q_2$. □

Given this result, the relative containment result follows:

THEOREM 5.3. *Given positive queries Q_1 and Q_2 and conjunctive views $\mathcal{V}$, where Q_2 and $\mathcal{V}$ may contain arbitrary comparison predicates, the problem of determining whether $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$ is in Π_2^P.*

Proof sketch: By Theorem 5.2, we can decide whether $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$ by building P_1, the maximally-contained query plan for Q_1, and checking whether $P_1^{exp} \sqsubseteq Q_2$. Suppose Q_1 contains a total of n subgoals in all its rules. Since Q_1 is nonrecursive and does not contain comparison predicates, it is still the case that each conjunctive query plan for Q_1 need only contain at most n subgoals. Therefore, we can prove that $P_1^{exp} \not\sqsubseteq Q_2$, and hence that $Q_1 \not\sqsubseteq_{\mathcal{V}} Q_2$, by showing the following: There exists a conjunctive query plan CQ with at most n subgoals such that $CQ^{exp} \sqsubseteq Q_1$ but $CQ^{exp} \not\sqsubseteq Q_2$.
To prove that determining if $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$ is in Π_2^P, we argue that the above algorithm for showing $Q_1 \not\sqsubseteq_{\mathcal{V}} Q_2$ is in Σ_2^P. Since Q_1 does not contain comparison predicates, checking that $CQ^{exp} \sqsubseteq Q_1$ is in NP. Since checking containment of a conjunctive query with inequalities in a positive query with inequalities is in Π_2^P [28; 39], checking non-containment of such queries is in Σ_2^P. Therefore, the entire procedure above is in Σ_2^P. □

Because of the lower bound proved in Section 3 for relative containment of conjunctive queries and views without comparison predicates, the bounds in Theorems 5.1 and 5.3 are tight.

6. CONCLUSIONS AND OPEN PROBLEMS

In this paper, we have introduced the notion of relative containment, which formalizes query containment in the context of the data integration framework. First, we provided tight complexity bounds for the general problem, with positive queries and conjunctive views. We then considered binding pattern adornments on the view predicates. There we showed that although the access limitations may require a recursive query plan to retrieve all certain answers to each query, even when the queries are conjunctive, relative containment is still decidable. Finally, we gave tight complexity bounds on restricted cases of relative containment with comparison predicates in the queries and views.

Several open problems remain, notably in cases where it is known that even producing the certain answers is co-NP-hard in the size of the view instances. These include the

case when both queries can contain arbitrary comparison predicates, as well as when the sources are assumed to be complete (i.e. the closed-world assumption) [1].
As the following example shows, considering sources to be complete does affect relative containment:

EXAMPLE 5. *Consider the following queries and views:*

$$q_1(x,y) \text{ :- } p(x,y). \qquad v_1(x) \text{ :- } p(x,y).$$
$$q_2(x,y) \text{ :- } r(x,y). \qquad v_2(y) \text{ :- } p(x,y).$$
$$v_3(x,y) \text{ :- } p(x,y), r(x,y).$$

Under the assumption of incomplete sources, $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$. In particular, views v_1 and v_2 don't provide any certain answers to q_1. For example, if $v_1(a)$ is true, we can infer that there is some constant c such that $p(a,c)$ is true. However, since sources are potentially incomplete, it is impossible to learn what this constant c is.
On the other hand, under the assumption of complete sources, consider the view instance $I = \{v_1(a), v_2(b)\}$. Since v_1 and v_2 are complete, it must be the case that $p(a,b)$ is true, so (a,b) is a certain answer of Q_1. However, Q_2 has no certain answers, so $Q_1 \not\sqsubseteq_{\mathcal{V}} Q_2$.

In this paper, we have considered relative containment for data integration systems that use local-as-view source descriptions. As we pointed out earlier, relative containment algorithms and complexity results are straightforward for the global-as-view case. An open problem is to consider relative containment in the third approach to specifying source descriptions, based on description logics [7; 6; 8].

7. REFERENCES

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Seattle, WA, 1998.

[2] S. Adali, K. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Montreal, Canada, 1996.

[3] A. Aho, Y. Sagiv, and J. D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, (8)2:218–246, 1979.

[4] J. L. Ambite, N. Ashish, G. Barish, C. A. Knoblock, S. Minton, P. J. Modi, I. Muslea, A. Philpot, and S. Tejada. ARIADNE: A system for constructing mediators for internet sources (system demonstration). In *Proc. of ACM SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[5] C. Beeri, G. Elber, T. Milo, Y. Sagiv, O.Shmueli, N.Tishby, Y.Kogan, D.Konopnicki, P. Mogilevski, and N.Slonim. Websuite-a tool suite for harnessing web data. In *Proceedings of the International Workshop on the Web and Databases*, Valencia, Spain, 1998.

[6] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Tucson, Arizona., 1997.

[7] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Answering queries using views in description logics. In *Working notes of the KRDB Workshop*, 1999.

[8] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 1993.

[9] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[10] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, Taipei, Taiwan, 1995.

[11] S. Chaudhuri and M. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 55–66, San Diego, CA., 1992.

[12] C. Chen and N. Roussopoulos. Implementation and performance evaluation of the ADMS query optimizer. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, March 1994.

[13] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[14] S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 330–341, 1996.

[15] O. Duschka, M. Genesereth, and A. Levy. Recursive query plans for data integration. *Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems*, 1999.

[16] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Tucson, Arizona., 1997.

[17] O. M. Duschka and M. R. Genesereth. Query planning in infomaster. In *Proceedings of the ACM Symposium on Applied Computing*, San Jose, CA, 1997.

[18] D. Florescu, L. Raschid, and P. Valduriez. A methodology for query reformulation in cis using semantic knowledge. *Int. Journal of Intelligent & Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems*, 5(4), 1996.

[19] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of the National Conference on Artificial Intelligence*, 1999.

[20] M. Friedman and D. Weld. Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan*, 1997.

[21] M. T. Friedman. *Optimization Issues in Data Integration*. PhD thesis, University of Washington, 1999.

[22] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132, March 1997.

[23] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of the Int. Conf. on Database Theory (ICDT)*, 1999.

[24] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 45–55, Minneapolis, Minnesota, 1994.

[25] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, Athens, Greece, 1997.

[26] Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution engine for data integration. In *Proc. of ACM SIGMOD Conf. on Management of Data*, 1999.

[27] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal*, 5(1):35–47, 1996.

[28] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160, 1988.

[29] C. T. Kwok and D. S. Weld. Planning to gather information. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[30] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1204–1210, 1999.

[31] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, San Jose, CA, 1995.

[32] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, Bombay, India, 1996.

[33] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 171–181, Dublin, Ireland, 1993.

[34] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, San Jose, CA, 1995.

[35] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1981.

[36] O. Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[37] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

[38] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118, 1996.

[39] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 331–345, San Diego, CA., 1992.

[40] H. Z. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 245–254, Brighton, England, 1987.