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 \( t \) is a certain answer to a query \( Q \) over the mediated schema with respect to a set of source instances if \( t \) 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.

\[\text{CarDesc}(\text{CarNo}, \text{Model}, \text{Color}, \text{Year})\]

\[\text{Review}(\text{Model}, \text{Review}, \text{Rating})\]

\text{CarDesc} describes cars for sale, including their number, model, color and year of manufacture. \text{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, \text{RedCars}, provides listings of red cars, while the second source,
Consider the following three queries:

\[ Q_1 : q_1(CarNo, Review) :- \]
\[ \text{CarDesc}(CarNo, Model, C, Y), \]
\[ \text{Review}(Model, Review, Rating). \]

\[ Q_2 : q_2(CarNo, Review) :- \]
\[ \text{CarDesc}(CarNo, Model, C, Y), \]
\[ \text{Review}(Model, Review, 10). \]

\[ Q_3 : q_3(CarNo, Review) :- \]
\[ \text{CarDesc}(CarNo, Model, C, Y), \]
\[ \text{Review}(Model, Review, 10), Y < 1990. \]

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 \( (\text{Rate} = 10) \) 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, \( Q_2 \) 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 predicates (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 \( \bar{X} = \emptyset \).

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 \( \neq, <, >, \leq, \geq \), 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 relation 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 \( V = V_1, \ldots, V_n \) described as views, and instances \( I = v_1, \ldots, v_n \) for the sources, the tuple \( t \) is a certain answer to \( Q \) w.r.t. \( I \) if \( t \) is in \( Q(D) \) for any database \( D \) such that \( I \subseteq V(D) \). We denote the set of certain answers of \( Q \) given \( I \) by \( \text{cert}(Q, I) \).

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 \( V \), the expansion \( P^\text{exp} \) of \( P \) is obtained from \( P \) by replacing all source relations in \( P \) with the bodies of the corresponding views in \( V \), 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 \( V \) if \( P^\text{exp} \subseteq Q \) and for every query plan \( P' \) such that \( P'^\text{exp} \subseteq Q \), \( P' \subseteq P \).

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 \( P_1 \):

\[
p_1(\text{CarNo, Review}) :-
   \text{CarDesc}(\text{CarNo, Model, C, Y}),
   \text{Review}(\text{Model, Review, Rating}).
\]

\[
\text{CarDesc}(\text{CarNo, Model, red, Year}) :-
   \text{RedCars}(\text{CarNo, Model, Year}).
\]

\[
\text{CarDesc}(\text{CarNo, Model, f(\text{CarNo, Model, Year}), Year}) :-
   \text{AntiqueCars}(\text{CarNo, Model, Year}).
\]

\[
\text{Review}(\text{Model, Review, 10}) :-
   \text{CarAndDriver}(\text{Model, Review}).
\]

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 \( P'_1 \):

\[
p'_1(\text{CarNo, Review}) :-
   \text{RedCars}(\text{CarNo, Model, Year}).
\]

\[
\text{CarAndDriver}(\text{Model, Review}).
\]

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 \( V = V_1, \ldots, V_n \) and queries \( Q_1 \) and \( Q_2 \), we say that \( Q_1 \) is contained in \( Q_2 \) relative to \( V \), denoted by \( Q_1 \subseteq_{V} Q_2 \).
$Q_2$, if for any instance $I$ of the views $V$, 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 $V$ 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 $Q_1 \sqsubseteq_V Q_2$ is to construct $P_1$ and $P_2$ and check whether $P_1 \subseteq P_2$. 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 $V$ of conjunctive views, determining whether $Q_1 \sqsubseteq_V Q_2$ is in $\Pi^P_2$.

**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 $CQ_1 \sqsubseteq P_1$, 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 $P_1 \subseteq P_2$, we check whether for every conjunctive query plan $CQ_1$ with at most $n$ subgoals such that $CQ_1 \sqsubseteq P_1$, there exists a conjunctive query plan $CQ_2$ with at most $m$ subgoals such that $CQ_2 \sqsubseteq P_2$ and $CQ_1 \sqsubseteq CQ_2$. Since deciding containment of a conjunctive query in a positive query is in $NP$ [9; 35], the time complexity of this algorithm for each $CQ_1$ is in $NP$, so the total time is in $\Pi^P_2$. □

**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 $V$ of conjunctive views, it is decidable whether $Q_1 \sqsubseteq_V Q_2$.

**Proof:** As mentioned above, we can construct datalog query plans $P_1$ and $P_2$ such that $Q_1 \sqsubseteq_V Q_2 \iff P_1 \subseteq P_2$. 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 $P_1 \subseteq P_2$ [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 $V$ of conjunctive views, determining whether $Q_1 \sqsubseteq_V Q_2$ is $\Pi^P_2$-hard.

**Proof sketch:** We reduce the $\forall \exists$-CNF problem, known to be $\Pi^P_2$-complete [37], to our problem. The $\forall \exists$-CNF problem is defined as follows: Given a 3-CNF formula $F$ with variables $\mathcal{X}$ and $\mathcal{Y}$, is it the case that for each truth assignment to $\mathcal{Y}$, there exists a truth assignment to $\mathcal{X}$ that satisfies $F$?

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

$$\mathcal{X} = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\}$$

and clauses $\mathcal{C} = \{c_1, \ldots, c_p\}$. 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 $r_1, \ldots, r_p$ of arity 3. Each predicate represents a different clause in $F$. Query $Q'_1$ simply records which variables are in each clause. It is defined as follows:

$$Q'_1() :- r_1(z_{1,1}, z_{1,2}, z_{1,3}), \ldots, r_p(z_{p,1}, z_{p,2}, z_{p,3}).$$

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 of the $\{a_{i,1}, a_{i,2}, a_{i,3}\}$ 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 $(x_1 \lor x_2 \lor y_1) \land \lnot x_1 \lor \lnot x_2 \lor \lnot y_2$. We build the following two queries:

$$Q'_1() :- r_1(x_1, x_2, y_1), r_2(x_1, x_2, y_2).$$

$$Q'_2() :- r_1(1, 1, 1), r_1(1, 1, 0), r_1(1, 0, 1), r_1(1, 0, 0), r_1(0, 1, 1), r_1(0, 1, 0), r_1(0, 0, 1), r_1(0, 0, 0), r_2(0, 1, 1), r_2(0, 1, 0), r_2(1, 0, 1), r_2(1, 0, 0), r_2(1, 1, 0).$$

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 $c_1(y_1), \ldots, c_m(y_m)$, where each $c_i$ is a new EDB predicate. To create $Q_2$ we add to $Q'_2$ the new subgoals $c_1(u_1), \ldots, c_m(u_m)$, 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: $v_i(z_{i,1}, z_{i,2}, z_{i,3}) :- r_i(z_{i,1}, z_{i,2}, z_{i,3})$. Finally, for each variable $y_i$, we have two views that represent that variable being assigned the truth value zero and one respectively: $w_{i,0}() :- e_i(0)$ and $w_{i,1}() :- e_i(1)$.

For our example formula, $(x_1 \lor x_2 \lor y_1) \land \lnot x_1 \lor \lnot x_2 \lor \lnot y_2)$,
we build the following queries and views:

\[ q_1() \triangleq r_1(x_1, x_2, y_1), r_2(x_1, x_2, y_2), e_1(y_1), e_2(y_2). \]

\[ q_2() \triangleq r_1(1, 1, 1), r_1(1, 1, 0), r_1(1, 0, 1), r_1(1, 0, 0), r_1(0, 1, 1), r_1(0, 1, 0), r_1(0, 0, 1), r_1(0, 0, 0), r_2(0, 0, 1), r_2(0, 1, 0), r_2(1, 0, 1), r_2(1, 0, 0), r_1(1, 1, 1), r_2(1, 1, 0), e_1(u_1), e_2(u_2). \]

\[ v_1(z_1, z_2, z_3) \triangleq r_1(z_1, z_2, z_3). \]

\[ v_2(z_1, z_2, z_3) \triangleq r_2(z_1, z_2, z_3). \]

\[ w_{1,0}() \triangleq e_1(0). \]

\[ w_{1,1}() \triangleq e_1(1). \]

\[ w_{2,0}() \triangleq e_2(0). \]

\[ w_{2,1}() \triangleq e_2(1). \]

We now show that \( F \) is \( \forall \exists \)-satisfiable if and only if \( Q_2 \cup Q_1 \). Let \( P_r (P_2) \) be the query plan that produces all the necessary answers to \( Q_1 (Q_2) \), constructed by the inverse rules algorithm. In particular, \( P_r \) consists of the union of all the obtained views and the inverted view definitions, and similarly for \( P_2 \). In our example, the inverted view definitions are as follows:

\[ r_1(z_1, z_2, z_3) \triangleq v_1(z_1, z_2, z_3). \]

\[ r_2(z_1, z_2, z_3) \triangleq v_2(z_1, z_2, z_3). \]

\[ e_1(0) \triangleq w_{1,0}(). \]

\[ e_1(1) \triangleq w_{1,1}(). \]

\[ e_2(0) \triangleq w_{2,0}(). \]

\[ e_2(1) \triangleq w_{2,1}(). \]

We know that \( Q_2 \cup Q_1 \) is equivalent to \( P_r \). Since the queries are conjunctive, \( P_r \) and \( P_2 \) are nonrecursive datalog programs. Further, by construction of the queries and views, \( P_r \) and \( P_2 \) do not contain function terms. Therefore, we can unfold \( P_r (P_2) \) into an equivalent union of conjunctive queries \( P_1 \) (\( P_2 \)). Therefore, \( P_2 \subseteq P_1 \) is equivalent to \( P_2 \subseteq P_1 \).

It is easy to see that \( P_1 \) and \( P_2 \) are each a union of \( 2^n \) conjunctive queries, one per truth assignment to \( \mathcal{F} \). In particular, the conjunctive query \( CQ_1^F \) in \( P_1 \), corresponding to truth assignment \( A = \{ a_1, \ldots, a_m \} \) to \( \mathcal{F} \), 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 \( CQ_2^F \) in \( P_2 \), corresponding to truth assignment \( A = \{ a_1, \ldots, a_m \} \) to \( \mathcal{F} \), 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:

\[ CQ_1^F () \triangleq v_1(x_1, x_2, 1), v_2(x_1, x_2, 0), w_{1,1}(), w_{2,0}(). \]

\[ CQ_2^F () \triangleq v_1(1, 1, 1), v_1(1, 0, 1), v_1(0, 1, 0), v_1(0, 0, 1), v_1(0, 0, 0), v_2(0, 0, 1), v_2(0, 1, 0), v_2(1, 0, 0), v_2(1, 0, 0), v_2(1, 0, 0), w_{1,1}(), w_{2,0}(). \]

By \[35\], \( P_2 \subseteq P_1 \) if and only if each conjunctive query in \( P_2 \) is contained in some conjunctive query in \( P_1 \). Consider the conjunctive query \( CQ_2^F \) in \( P_2 \), corresponding to some truth assignment \( A = \{ a_1, \ldots, a_m \} \) to \( \mathcal{F} \). We know that \( CQ_2^F \) 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 \( CQ_1^F \), the conjunctive query in \( P_1 \), corresponding to the same truth assignment \( A \) to \( \mathcal{F} \). Therefore, we have that \( P_2 \subseteq P_1 \) if and only if each conjunctive query in \( P_2 \) is \( \forall \exists \)-equivalent to \( CQ_1^F \).

Let \( CQ_2^A \) be \( CQ_2^F \) with all of the \( w \) subgoals removed from its body, and similarly for \( CQ_1^A \). Since for each assignment \( A \) to \( \mathcal{F} \), \( CQ_2^A \) and \( CQ_1^A \) have the same \( w \) subgoals in their bodies, we have \( CQ_2^A \subseteq CQ_1^A \) if and only if \( CQ_2^A \subseteq CQ_1^A \) is \( \forall \exists \)-equivalent to \( CQ_1^A \) if and only if there is a truth assignment \( A' \) to \( \mathcal{F} \) such that \( A' \cup A \) satisfies \( F \). By definition, this is true if and only if \( F \) is \( \forall \exists \)-satisfiable.

The combination of Theorems 3.1 and 3.3 implies that relative containment for conjunctive queries and views is \( \Pi_2 \)-complete. In contrast, ordinary conjunctive query containment is \( NP \)-complete [9]. The theorems also imply \( \Pi_2 \)-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 \( \text{RedCars} \) indicates that the model of the car needs to be provided in order to obtain the cars for sale of that model.

\[ \text{RedCars}^{bf}(\text{CarNo, Model, Year}) \subseteq \text{CarDescription}^{bf}(\text{CarNo, Model, red, Year}). \]

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 \( r(X) \) in its body, for each position \( i \) of \( r \)'s adornment that is a \( b \), either the \( i \)th parameter of \( r(X) \) 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:

\[ Q : q(CarNo, Year) : - \]
\[ CarDescription(CarNo, Model, red, Year). \]

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:

\[ p(CarNo, Year) : - \]
\[ RedCars(CarNo, corolla, Year). \]

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 \( V \), and a set \( B \) of one binding pattern adornment for each view predicate, a query plan \( P \) is sound relative to \( Q \), \( V \), and \( B \) if \( P \) is executable, the constants in \( P \) are a subset of those in \( Q \cup V \), and \( P^{\exp} \subseteq Q \).

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 \( V = V_1, \ldots, V_n \), a set \( B \) of one binding pattern adornment for each view predicate, and instances \( I = i_1, \ldots, i_n \) 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 \( Q \), \( V \), 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 \( V \) and a set \( B \) of one binding pattern adornment for each view predicate if \( P \) is sound relative to \( Q \), \( V \), and \( B \), and for every query plan \( P' \) that is sound relative to \( Q \), \( V \), and \( B \), \( P' \subseteq P \).

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 \( V = V_1, \ldots, V_n \), a set \( B \) of one binding pattern adornment for each view predicate, and queries \( Q_1 \) and \( Q_2 \) such that \( Q_1 \cup \bigcup V \) has a subset of the constants in \( Q_2 \cup V \), we say that \( Q_1 \) is contained in \( Q_2 \) relative to \( V \) and the binding patterns, denoted by \( Q_1 \subseteq_{V,B} Q_2 \), 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 \( Q_1 \subseteq_{V,B} Q_2 \iff P_1 \subseteq P_2 \), 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 \( V \) 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 \( Q_1 \cup \bigcup V \) has a subset of the constants in \( Q_2 \cup V \). Let \( P_1 \) and \( P_2 \) be the maximally-contained query plans of \( Q_1 \) and \( Q_2 \) using \( V \) and \( B \). Then, \( P_1 \subseteq P_2 \iff P_1^{\exp} \subseteq P_2^{\exp} \).

**Proof:** For the “\( \Rightarrow \)” direction, \( P_1 \subseteq P_2 \iff \forall I. P_1(I) \subseteq P_2(I) \Rightarrow \forall D. P_1(V(D)) \subseteq P_2(V(D)) \iff P_1^{\exp} \subseteq P_2^{\exp} \). Since \( P_2 \) is the maximally-contained query plan for \( Q_2 \) using \( V \) and \( B \), \( P_2^{\exp} \subseteq Q_2 \), so transitively we have \( P_1^{\exp} \subseteq Q_2 \). For the “\( \Leftarrow \)” direction, suppose \( P_1^{\exp} \subseteq Q_2 \). Since \( P_1 \) is the maximally-contained query plan for \( Q_1 \) using \( V \) and \( B \), \( P_1 \) is executable and has a subset of the constants in \( Q_1 \cup V \). We are given that \( Q_1 \cup V \) has a subset of the constants in \( Q_2 \cup V \), so transitively \( P_1 \) does as well. Therefore, \( P_1 \) is sound relative to \( Q_2 \), \( V \), and \( B \). Since \( P_2 \) is the maximally-contained query plan for \( Q_2 \) and \( V \), by the definition of maximal containment we have \( P_1 \subseteq P_2 \).

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 \( V \) of conjunctive views, and a set \( B \) of one binding pattern adornment for each view predicate, determining whether \( Q_1 \subseteq_{V,B} Q_2 \) 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 \( \theta \) is either \(<\) or \(\leq\) (alternatively, \( \theta \) is either \(\geq\) or \(>\)). Therefore, we can show the following:

**Theorem 5.1.** Given positive queries \( Q_1 \) and \( Q_2 \) and conjunctive views \( V \), where all queries and views may contain semi-interval constraints, the problem of determining whether \( Q_1 \sqsubseteq_V Q_2 \) is in \( \Pi^2_2 \).

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:

\[ p_3(Car\_No, Review) :- \]
\[ RedCars(Car\_No, Model, Year), \]
\[ Car\_And\_Driver(Model, Review), Year < 1990. \]
\[ p_4(Car\_No, Review) :- \]
\[ AntiqueCars(Car\_No, Model, Year), \]
\[ Car\_And\_Driver(Model, Review). \]

Because \( P_3 \) does not contain plan \( P_1^P \) 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 \( V \), 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 \( V \), where \( Q_2 \) and \( V \) may contain arbitrary comparison predicates, let \( P_1 \) be the maximally-contained query plan for \( Q_1 \). Then \( Q_1 \sqsubseteq_V Q_2 \iff P_1^{exp} \subseteq Q_2 \).

**Proof:** For the “\( \Rightarrow \)” direction,

\[ Q_1 \sqsubseteq_V Q_2 \]
\[ \forall I. \text{certain}(Q_1, I) \subseteq \text{certain}(Q_2, I) \]
\[ \forall I. P_I(I) \subseteq \text{certain}(Q_2, I) \]
\[ \forall D. P_D(V(D)) \subseteq \text{certain}(Q_2, V(D)) \]
\[ \forall D. P_D(V(D)) \subseteq Q_2(D) \]
\[ P_1^{exp} \subseteq Q_2 \]

For the “\( \Leftarrow \)” direction, we prove the contrapositive. Suppose \( Q_1 \not\sqsubseteq_V Q_2 \). Then there exists an instance \( I \) of the views and a tuple \( t \) such that \( t \in \text{certain}(Q_1, I) \) but \( t \not\in \text{certain}(Q_2, I) \).

Given this result, the relative containment result follows:

**Theorem 5.3.** Given positive queries \( Q_1 \) and \( Q_2 \) and conjunctive views \( V \), where \( Q_2 \) and \( V \) may contain arbitrary comparison predicates, the problem of determining whether \( Q_1 \sqsubseteq_V Q_2 \) is in \( \Pi^2_2 \).

**Proof sketch:** By Theorem 5.2, we can decide whether \( Q_1 \sqsubseteq_V Q_2 \) by building \( P_1 \), the maximally-contained query plan for \( Q_1 \), and checking whether \( P_1^{exp} \subseteq Q_2 \). 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 \( P_1^{exp} \not\subseteq Q_2 \), and hence that \( Q_1 \not\sqsubseteq_V Q_2 \), by showing the following: There exists a conjunctive query plan \( CQ \) with at most \( n \) subgoals such that \( CQ^{exp} \subseteq Q_1 \) but \( CQ^{exp} \not\subseteq Q_2 \). To prove that determining if \( Q_1 \sqsubseteq_V Q_2 \) in \( \Pi^2_2 \), we argue that the above algorithm for showing \( Q_1 \sqsubseteq_V Q_2 \) is in \( \Sigma^P_2 \). Since \( Q_1 \) does not contain comparison predicates, checking that \( CQ^{exp} \subseteq Q_1 \) is in \( NP \). Since checking containment of a conjunctive query with inequalities in a positive query with inequalities is in \( \Pi^2_2 \) [28; 39], checking non-containment of such queries is in \( \Sigma^P_2 \). Therefore, the entire procedure above is in \( \Sigma^P_2 \). □

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:**

\[
q_1(x, y) \leftarrow p(x, y).
\]
\[
q_2(x, y) \leftarrow r(x, y).
\]
\[
v_1(x) \leftarrow p(x, y).
\]
\[
v_2(y) \leftarrow p(x, y).
\]
\[
v_3(x, y) \leftarrow p(x, y), r(x, y).
\]

*Under the assumption of incomplete sources, \( Q_1 \subseteq_v Q_2 \). In particular, \( v_1 \) 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 \( Q_1 \not\subseteq_v Q_2 \).*

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


