Available online at www.sciencedirect.com

SCIENCE DIRECT®

ACADEMIC PRESS

Journal of Computer and System Sciences 66 (2003) 20–39

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

http://www.elsevier.com/locate/jcss

Query containment for data integration systems☆

Todd Millstein,* Alon Halevy, and Marc Friedman[1]

Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98195-2350, USA

Received 1 October 2000; revised 1 January 2002

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 data-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.
© 2003 Published by Elsevier Science (USA).

Keywords: Query containment; Data integration; Query rewriting; Views; Datalog; Binding patterns

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

☆This research was funded by National Science Foundation Grant IIS-9978567 and by a Sloan Fellowship. An earlier version of this paper appeared in PODS 2000 [37].

*Corresponding author.

E-mail addresses: todd@cs.washington.edu (T. Millstein), alon@cs.washington.edu (A. Halevy), friedman@cs.washington.edu (M. Friedman).

[1]Currently at Viathan Corporation, Seattle, Washington, USA.

their queries, 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 [3,5,6,14,19,20,22,24,27,28,34] as well as several industrial solutions (e.g., [16]).

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 are 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 schemas.

One of the common approaches to specifying source descriptions, known as the *local-as-view* (or *source-centric*) approach, is to describe data sources as containing answers to views over the mediated schema [19,22,32,34,38]. In a second approach, known as *global-as-view* (or *query-centric*), the mediated schema is described as containing answers to views over the source relations [3,24,27]. The local-as-view approach allows new sources to be added and removed modularly, while the global-as-view approach requires source descriptions to be modified when such changes occur. On the other hand, query answering is straightforward in the global-as-view approach, where the answers can be obtained by simply composing the query with the views, while the local-as-view approach requires a more sophisticated form of query rewriting. In this paper we consider data-integration systems where the source descriptions are provided using the local-as-view approach.

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 of 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 [11,33,43,45], while in others, specialized algorithms are necessary [1,25].

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 [4,10,40], detecting independence of queries from database updates [35], rewriting queries using views [33,43], maintenance of integrity constraints [26], and semantic data caching [3,13,15,29].

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 set of instances 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 1.1. 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).

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

Consider the following three queries:

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

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, query Q_2 is contained in query Q_1 because Q_2 applies a stronger condition ($\mathit{Rating} = 10$) 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 always 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 are the same because the queries are equivalent, or because they are equivalent for the currently 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 of each query can be obtained by a nonrecursive 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 in which the queries are nonrecursive datalog programs.

Next, we consider the case where there are limitations on access patterns to the underlying sources. Here, it follows from [1,17] 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 [41]. 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.

As mentioned above, in the global-as-view approach, the set of certain answers of a query can be obtained by simply composing the query with the views. Therefore, algorithms and complexity results for relative containment in this case are straightforward corollaries of traditional query containment results.

2. Preliminaries

We begin by defining the terms used throughout the paper.

2.1. Queries

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

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

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 $\bar{X} = \emptyset$.

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 contains only EDB predicates. A nonrecursive datalog program 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 $j \leqslant i$. 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 $\neq, <, >, \leqslant$, and $\geqslant$, 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 relational subgoal in the body of the rule.

The *size* of a datalog rule is the number of subgoals in its body. The size of a datalog program is the sum of the sizes of its rules.

2.2. Query containment

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

Much is known about the complexity of query containment for subsets of datalog. Containment of conjunctive queries is NP-complete [10]. Containment of a conjunctive query in a nonrecursive query is also NP-complete, while containment of two nonrecursive queries is Π_2^P-complete [40]. Containment is decidable when at least one of the two queries is nonrecursive [12], while containment of arbitrary datalog programs is undecidable [41].

Recall that a language L is in NP iff L can be described as $\{w | \exists x.\phi(w,x)\}$, where x has size polynomial in the size of w and ϕ is computable in polynomial time. Similarly, a language L is in co-NP iff L can be described as $\{w | \forall x.\phi(w,x)\}$, where x has size polynomial in the size of w and ϕ is computable in polynomial time. Finally, a language L is in Π_2^P iff L can be described as $\{w | \forall x.\exists y.\phi(w,x,y)\}$, where x and y have sizes polynomial in the size of w and ϕ is computable in polynomial time [42].

2.3. 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).

2.4. 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 a set of instances $I = v_1, \ldots, v_n$ for the sources, the tuple $\bar{t}$ is a *certain answer* of 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 w.r.t. I by $certain(Q, I)$.

The complexity of finding certain answers is discussed in [1,25]. 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*, which is a datalog program whose EDB relations are the source relations.

Given a query plan P for Q w.r.t. $\mathcal{V}$, 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 [17].

The query plan that produces the most answers is called the maximally contained plan, and is formally defined as follows [17]:

Definition 2.2 (Maximally contained plan). A query plan P is *maximally contained* in a query Q w.r.t. views $\mathcal{V}$ if (1) $P^{\exp} \sqsubseteq Q$ and (2) for every query plan P' such that $P'^{\exp} \sqsubseteq Q$, it is the case that $P' \sqsubseteq P$.

It is shown in [1] that the maximally contained query plan computes the certain answers of the query:

Lemma 2.1. *Given a datalog query Q, a set of conjunctive views $\mathcal{V}$, and a query plan P that is maximally contained in Q w.r.t. $\mathcal{V}$, P produces exactly the certain answers of Q w.r.t. I, for each instance I of $\mathcal{V}$.*

The maximally contained query plan is obtained by an algorithm for rewriting queries using views [11,18,33,43,45]. In our discussion, we rely on the properties of one such algorithm, the *inverse rules* algorithm [17]. Informally, the algorithm simply "inverts" each view definition, producing several datalog rules, one per relational 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 2.1. The maximally contained query plan constructed by the inverse rules algorithm for query Q_1 from Example 1.1, given the three sources listed there, is the following plan P_1:

$$
\begin{array}{ll}
p_1(CarNo, Review){:}- & CarDesc(CarNo, Model, Color, Year), \\
 & Review(Model, Review, Rating). \\
CarDesc(CarNo, Model, red, Year){:}- & RedCars(CarNo, Model, Year). \\
CarDesc(CarNo, Model, f(CarNo, Model, Year), Year){:}- & AntiqueCars(CarNo, Model, Year). \\
Review(Model, Review, 10){:}- & CarAndDriver(Model, Review).
\end{array}
$$

Since Q_2 from Example 1.1 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 nonrecursive query is nonrecursive, and the maximally contained query plan of a recursive query is recursive [17]. As mentioned above and shown in Example 2.1, the query plan may contain function terms; [17] shows how to remove these function terms. This procedure results in a nonrecursive query plan if the original query plan was nonrecursive.

Example 2.2. Removing function terms from the query plan in Example 2.1 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 source descriptions and the query are conjunctive. Since datalog programs have polynomial data complexity [2], a query plan in datalog that produces all certain answers cannot always be found in these cases. Restricted cases exist where the query contains comparison predicates but the certain answers can be found in polynomial time [23].

2.5. 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 1.1, 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 $\mathcal{V}$, $certain(Q_1, I) \subseteq certain(Q_2, I)$.

2.6. Complexity measure

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, we measure 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 and contain no comparison predicates.

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 of 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 bound on the complexity of relative containment:

Theorem 3.1. *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 $Q_1 \sqsubseteq_{\mathcal{V}} Q_2 \Leftrightarrow P_1 \sqsubseteq P_2$. 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 $P_1 \sqsubseteq P_2$. □

We can provide a tighter upper bound for relative containment of two nonrecursive queries. First we state a variant of a lemma proven in [33].

Lemma 3.1. *Let Q be a nonrecursive datalog program of size n, $\mathcal{V}$ be a set of conjunctive views, and CP be a conjunctive query plan such that $CP^{\exp} \sqsubseteq Q$. Then there exists a conjunctive query plan CP_0 of size at most n such that $CP_0^{\exp} \sqsubseteq Q$ and $CP \sqsubseteq CP_0$.*

The above lemma can then be used to characterize the maximally contained query plan.

Lemma 3.2. *Given a nonrecursive datalog program Q of size n and a set $\mathcal{V}$ of conjunctive views, the maximally contained query plan P in Q w.r.t $\mathcal{V}$ is equivalent to the union of every conjunctive query plan CP of size at most n such that $CP^{\exp} \sqsubseteq Q$.*

Proof. Let the union of every conjunctive query plan CP of size at most n such that $CP^{\exp} \sqsubseteq Q$ be P_0. We will show that $P_0 \sqsubseteq P$ and $P \sqsubseteq P_0$.

$P_0 \sqsubseteq P$: We are given that every conjunctive query CP in P_0 has the property that $CP^{\exp} \sqsubseteq Q$. This means that $P_0^{\exp} \sqsubseteq Q$ as well. Since P is the maximally contained query plan, by Definition 2.2 we have $P_0 \sqsubseteq P$.

$P \sqsubseteq P_0$: Consider a tuple $\bar{t} \in P(D)$, for some database D. We will show that $\bar{t} \in P_0(D)$ as well. Since Q is nonrecursive, so is P, so P is equivalent to a finite union of conjunctive queries. Further, one of these conjunctive queries, call it CP, must produce $\bar{t}$ given D. Since P is the maximally contained query plan, by Definition 2.2 we have $P^{\exp} \sqsubseteq Q$, so it is also the case that $CP^{\exp} \sqsubseteq Q$. Therefore by Lemma 3.1, there must exist some query plan CP_0 that has size at most n such that

$CP_0^{\exp} \sqsubseteq Q$ and $CP \sqsubseteq CP_0$ [33]. Since $\bar{t} \in CP$, also $\bar{t} \in CP_0$. Since CP_0 has size at most n and $CP_0^{\exp} \sqsubseteq Q$, CP_0 is in P_0, so $\bar{t} \in P_0(D)$. □

Finally, the above characterization of the maximally contained query plan provides an algorithm for checking relative containment.

Theorem 3.2. *Given two nonrecursive datalog programs Q_1 and Q_2 and a set $\mathscr{V}$ of conjunctive views, determining whether $Q_1 \sqsubseteq_{\mathscr{V}} Q_2$ is in Π_2^P.*

Proof. Suppose that Q_1 has size n and Q_2 has size m. Let P_1 (P_2) be the maximally contained query plan in Q_1 (Q_2) w.r.t $\mathscr{V}$. By [40], $P_1 \sqsubseteq P_2$ if and only if each conjunctive query in the unfolding of P_1 is contained in some conjunctive query in the unfolding of P_2. Therefore by Lemma 3.2, to decide whether $P_1 \sqsubseteq P_2$, it suffices to check whether for every conjunctive query plan CP_1 of size at most n, either ${CP_1}^{\exp} \not\sqsubseteq Q_1$ or there exists a conjunctive query plan CP_2 with size at most m such that ${CP_2}^{\exp} \sqsubseteq Q_2$ and $CP_1 \sqsubseteq CP_2$.

Since deciding containment of a conjunctive query in a nonrecursive query is in NP (hence noncontainment is in co-NP), the check described above has the form

$$\forall CP_1.(\forall u.\phi_1(Q_1, \mathscr{V}, CP_1, u) \vee \exists CP_2.(\exists v.\phi_2(Q_2, \mathscr{V}, CP_2, v) \wedge \exists z.\phi_3(CP_1, CP_2, z)))$$

where CP_1, CP_2, u, v, and z have sizes polynomial in the sum of the sizes of Q_1, Q_2, and $\mathscr{V}$, and ϕ_1, ϕ_2, and ϕ_3 are polynomial-time computable. By pulling all the quantifiers out front, we see that this check has the form $\forall x.\exists y.\phi(w, x, y)$, where x and y have sizes polynomial in the size of w and ϕ is computable in polynomial time. Therefore, the algorithm is in Π_2^P. □

3.2. Lower bounds

In this section, we show that the upper bound of Theorem 3.2 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 $\mathscr{V}$ of conjunctive views, determining whether $Q_1 \sqsubseteq_{\mathscr{V}} Q_2$ is Π_2^P-hard.*

Proof. We reduce the ∀∃-CNF problem, known to be Π_2^P-complete [42], to our problem. The ∀∃-CNF problem is defined as follows: Given a 3-CNF propositional formula F with variables $\bar{x}$ and $\bar{y}$, is it the case that for each truth assignment to $\bar{y}$, there exists a truth assignment to $\bar{x}$ that satisfies F?

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

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

and clauses $\bar{c} = \{c_1, \dots, c_p\}$. 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 [4] to show NP-hardness of conjunctive query containment. We use the EDB predicates $r_1, \dots, r_p$ of arity three, each predicate r_i representing clause c_i 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:

$$\begin{aligned}
&q_1'() \text{:- } r_1(x_1, x_2, y_1), r_2(x_1, x_2, y_2).\\
&q_2'() \text{:- } r_1(1,1,1), r_1(1,1,0), r_1(1,0,1), r_1(1,0,0), r_1(0,1,1), r_1(0,1,0), r_1(0,0,1),\\
&\qquad r_2(0,0,0), r_2(0,0,1), r_2(0,1,0), r_2(0,1,1), r_2(1,0,0), r_2(1,0,1), r_2(1,1,0).
\end{aligned}$$

Similar to the argument in [4], 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 each e_i 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:

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

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 of 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:

$r_1(z_1, z_2, z_3)$:- $v_1(z_1, z_2, z_3)$.
$r_2(z_1, z_2, z_3)$:- $v_2(z_1, z_2, z_3)$.
$e_1(0)$:- $w_{1,0}()$.
$e_1(1)$:- $w_{1,1}()$.
$e_2(0)$:- $w_{2,0}()$.
$e_2(1)$:- $w_{2,1}()$.

We know that $Q_2 \sqsubseteq_{\mathscr{V}} Q_1 \Leftrightarrow P_2 \sqsubseteq P_1$. 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, $P_2 \sqsubseteq P_1 \Leftrightarrow P_2* \sqsubseteq P_1*$.

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 $\bar{y}$. In particular, the conjunctive query CP_1^A in P_1*, corresponding to truth assignment $A = \{a_1, \ldots, a_m\}$ to $\bar{y}$, 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 CP_2^A in P_2*, corresponding to truth assignment $A = \{a_1, \ldots, a_m\}$ to $\bar{y}$, 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 $e_i(u_i)$ 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:

$cq_1^A()$:- $v_1(x_1, x_2, 1), v_2(x_1, x_2, 0), w_{1,1}(), w_{2,0}()$.
$cq_2^A()$:- $v_1(1,1,1), v_1(1,1,0), v_1(1,0,1), v_1(1,0,0), v_1(0,1,1), v_1(0,1,0),$
$v_1(0,0,1), v_2(0,0,0), v_2(0,0,1), v_2(0,1,0), v_2(0,1,1), v_2(1,0,0),$
$v_2(1,0,1), v_2(1,1,0), w_{1,1}(), w_{2,0}()$.

By [40], $P_2* \sqsubseteq P_1*$ if and only if each conjunctive query in P_2* is contained in some conjunctive query in P_1*. Consider the conjunctive query CP_2^A in P_2*, corresponding to some truth assignment $A = \{a_1, \ldots, a_m\}$ to $\bar{y}$. We know that CP_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 CP_1^A, the conjunctive query in P_1* corresponding to the same truth assignment A to $\bar{y}$. Therefore, we have that $P_2* \sqsubseteq P_1*$ if and only if for each assignment A to $\bar{y}$, $CP_2^A \sqsubseteq CP_1^A$.

Let CP_1^{A-} be CP_1^A with all of the w subgoals removed from its body, and similarly for CP_2^{A-}. Since for each assignment A to $\bar{y}$, CP_2^A and CP_1^A have the same w subgoals in their bodies, we have $CP_2^A \sqsubseteq CP_1^A \Leftrightarrow CP_2^{A-} \sqsubseteq CP_1^{A-}$. Note that CP_2^{A-} is identical to Q_2' above, and CP_1^{A-} is identical to Q_1', but with the $\bar{y}$ variables hard-coded to the truth assignment A. Thus, from the correctness of the reduction from CNF-satisfiability to conjunctive query containment described above, we know that for each assignment A to $\bar{y}$, $CP_2^{A-} \sqsubseteq CP_1^{A-}$ if and only if there exists a truth assignment A' to $\bar{x}$ such that $A' \cup A$ satisfies F. Therefore, $P_2* \sqsubseteq P_1*$ if and only if for each assignment A to $\bar{y}$, there exists a truth assignment A' to $\bar{x}$ such that $A' \cup A$ satisfies F. By definition, this is true if and only if F is $\forall\exists$-satisfiable. $\square$

Theorems 3.2 and 3.3 together imply that relative containment for conjunctive queries and views is Π_2^P-complete. In contrast, ordinary conjunctive query containment is only NP-complete [10]. The theorems also imply Π_2^P-completeness for relative containment when both queries are nonrecursive, matching the complexity of ordinary query containment in this case [40].

4. Binding pattern limitations

In this section we consider the common situation in which data sources 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 needs not be provided. For example, the adornment on *RedCars* in the following source description indicates that the model of the car needs to be provided in order to obtain the cars for sale of that model.

$$RedCars^{fbf}(CarNo, Model, Year) \subseteq CarDescription(CarNo, Model, red, Year).$$

We concentrate here on the case where each source has a single adornment. Sources with multiple possible access patterns can be modeled 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. We first define executable datalog programs, which are query plans that obey the binding pattern restrictions on sources.

Definition 4.1 (Executability). A datalog rule is *executable* if for each EDB subgoal $r(\bar{X})$ in its body, for each position i at which r's adornment contains a b, the ith parameter of $r(\bar{X})$ is either a constant or it is a variable that also appears in a subgoal to the left of $r(\bar{X})$ in the body of the rule. 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 can never 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 in some source instances, 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 $\mathcal{V}$, 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 (1) P is executable, (2) the constants in P are a subset of those in $Q \cup \mathcal{V}$, and (3) $P^{\exp} \sqsubseteq Q$.

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* of Q w.r.t. I and B if (1) $\bar{t}$ is a certain answer of Q w.r.t. I and (2) there exists a sound query plan P relative to $Q, \mathcal{V}$, and B such that $\bar{t} \in P(I)$.[2]

The problem of answering queries using views with binding pattern limitations is considered in [17,31,32,39]. The results of [17] imply that, when the query is in datalog and the views are conjunctive, a datalog maximally contained query plan can always be found. However, the maximally contained query plan is *recursive* in general, even when the query is conjunctive. This recursive 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 w.r.t. views $\mathcal{V}$ and a set B of one binding pattern adornment for each view predicate if (1) P is sound relative to $Q, \mathcal{V}$, and B and (2) for every query plan P' that is sound relative to $Q, \mathcal{V}$, and B, it is the case that $P' \sqsubseteq P$.

[2] When Q is monotonic, condition 1 follows from condition 2.

Analogous to Lemma 2.1, we can show that the maximally contained query plan in this setting computes the reachable certain answers of the query:

Lemma 4.1. *Given a datalog query Q, a set of conjunctive views $\mathcal{V}$, a set B of one binding pattern adornment for each view predicate, and a query plan P that is maximally contained in Q w.r.t. $\mathcal{V}$ and B, P produces exactly the reachable certain answers of Q w.r.t. I and B, for each instance I of $\mathcal{V}$.*

Proof. First we show that every element of $P(I)$ is a reachable certain answer. Consider some tuple $\bar{t} \in P(I)$. Since P is the maximally contained query plan, by Definition 4.4 P is sound relative to $Q, \mathcal{V}$, and B. Therefore by Definition 4.2 $P^{\exp} \sqsubseteq Q$. This means that $\forall D.P^{\exp}(D) \subseteq Q(D)$, so $\forall D.P(\mathcal{V}(D)) \subseteq Q(D)$. Since P is a datalog program, it is monotonic. Therefore, since $\bar{t} \in P(I)$ we have that $\bar{t} \in P(\mathcal{V}(D))$ for any database D such that $I \subseteq \mathcal{V}(D)$. Therefore, $\bar{t} \in Q(D)$ for any database D such that $I \subseteq \mathcal{V}(D)$, so by Definition 2.1 $\bar{t}$ is a certain answer of Q w.r.t. I. Since P is sound relative to $Q, \mathcal{V}$, and B and $\bar{t} \in P(I)$, by Definition 4.3 $\bar{t}$ is a reachable certain answer of Q w.r.t. I and B.

Next we show that every reachable certain answer is an element of $P(I)$. Suppose some reachable certain answer $\bar{t}$ is not in $P(I)$. By Definition 4.3, there is some query plan P' that is sound relative to $Q, \mathcal{V}$, and B such that $\bar{t} \in P'(I)$. Consider the query plan P_0 consisting of the union of P and P'. Since $\bar{t} \notin P(I)$ but $\bar{t} \in P'(I)$, we have $P_0 \not\sqsubseteq P$. Since P is the maximally contained query plan, by Definition 4.4 P is sound relative to $Q, \mathcal{V}$, and B. Therefore, P_0 is also sound relative to $Q, \mathcal{V}$, and B. Therefore again by Definition 4.4 we have $P_0 \sqsubseteq P$, so we have reached a contradiction.

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

Definition 4.5 (Relative containment with binding patterns). 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 w.r.t. $\mathcal{V}$ and B are a subset of the reachable certain answers of Q_2 w.r.t. $\mathcal{V}$ and B.

4.2. Decidability

As in Section 3, we know that $Q_1 \sqsubseteq_{\mathcal{V},B} Q_2 \Leftrightarrow P_1 \sqsubseteq P_2$, where P_1 (P_2) is the maximally contained query plan in Q_1 (Q_2) w.r.t. $\mathcal{V}$ and B. 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 [41].

However, the following lemma 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:

Lemma 4.2. *Let $\mathscr{V}$ 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 $Q_1 \cup \mathscr{V}$ has a subset of the constants in $Q_2 \cup \mathscr{V}$. Let P_1 and P_2 be the maximally contained query plans of Q_1 and Q_2 w.r.t. $\mathscr{V}$ and B. Then $P_1 \sqsubseteq P_2 \Leftrightarrow P_1^{\exp} \sqsubseteq Q_2$.*

Proof. For the "$\Rightarrow$" direction, $P_1 \sqsubseteq P_2 \Rightarrow \forall I.P_1(I) \subseteq P_2(I) \Rightarrow \forall D.P_1(\mathscr{V}(D)) \subseteq P_2(\mathscr{V}(D)) \Rightarrow \forall D.P_1^{\exp}(D) \subseteq P_2^{\exp}(D) \Rightarrow P_1^{\exp} \sqsubseteq P_2^{\exp}$. Since P_2 is the maximally contained query plan of Q_2 w.r.t. $\mathscr{V}$ and B, by Definition 4.4 we have $P_2^{\exp} \sqsubseteq Q_2$, so transitively we have $P_1^{\exp} \sqsubseteq Q_2$.

For the "$\Leftarrow$" direction, suppose $P_1^{\exp} \sqsubseteq Q_2$. Since P_1 is the maximally contained query plan for Q_1 w.r.t. $\mathscr{V}$ and B, by Definition 4.4 we have that P_1 is sound relative to $Q_1, \mathscr{V}$, and B. Therefore, by Definition 4.2 P_1 is executable and has a subset of the constants in $Q_1 \cup \mathscr{V}$. We are given that $Q_1 \cup \mathscr{V}$ has a subset of the constants in $Q_2 \cup \mathscr{V}$, so transitively P_1 does as well. Therefore by Definition 4.2 P_1 is sound relative to Q_2, $\mathscr{V}$, and B. Since P_2 is the maximally contained query plan for Q_2 and $\mathscr{V}$, by Definition 4.4 we have $P_1 \sqsubseteq P_2$. □

As the above lemma 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 nonrecursive one. Hence, the decidability results of [12] in this case entail the following theorem:

Theorem 4.1. *Given a (potentially recursive) datalog program Q_1, a nonrecursive datalog program Q_2, a set $\mathscr{V}$ of conjunctive views, and a set B of one binding pattern adornment for each view predicate, determining whether $Q_1 \sqsubseteq_{\mathscr{V},B} Q_2$ is decidable.*

Note that an analogous version of Lemma 4.2 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.

Recent work [36] has shown a decidability result for the global-as-view approach in the presence of binding patterns, for the case of conjunctive queries.

5. Comparison predicates

In this section, we study the complexity of relative containment when queries and views may contain comparison predicates of the form $\neq, <, >, \leqslant$, and $\geqslant$, interpreted over a dense domain.

In general, finding all certain answers of a query that contains comparison predicates is co-NP-hard in the size of the view instances [1,21], so datalog maximally contained query plans do not always exist. However, there are certain cases where it is known that datalog maximally contained query plans will exist [23]. One such case is when all comparison atoms are *semi-interval*. A comparison atom is a left (right) semi-interval constraint if it has the form $x\theta c$ ($c\theta x$), where x is a variable, c is a constant, and θ is either $<$ or $\leqslant$. Therefore, the following holds:

Theorem 5.1. *Given nonrecursive datalog queries Q_1 and Q_2 and conjunctive views $\mathscr{V}$, where all queries and views may contain only left (right) semi-interval constraints, the problem of determining whether $Q_1 \sqsubseteq_{\mathscr{V}} Q_2$ is in Π_2^P.*

The proof of the theorem follows from a parallel argument to that of Theorem 3.2. First, Lemmas 3.1 and 3.2 still hold, where the size of a query is now the number of relational subgoals in its body. Therefore, again we need to show that for every conjunctive query plan CQ_1 of size at most the size of Q_1, either $CQ_1^{\mathrm{exp}} \not\sqsubseteq Q_1$ or there exists a conjunctive query plan CQ_2 with size at most the size of Q_2 such that $CQ_2^{\mathrm{exp}} \sqsubseteq Q_2$ and $CQ_1 \sqsubseteq CQ_2$. Because the constraints are semi-interval, once the relational subgoals for a candidate CQ_1 (CQ_2) have been chosen, it is straightforward to add the appropriate comparison predicates such that $CQ_1^{\mathrm{exp}} \sqsubseteq Q_1$ ($CQ_2^{\mathrm{exp}} \sqsubseteq Q_2$) or to show that no such comparison predicates exist (since we only need to consider a finite number of constants in the comparison predicates). Since containment of a conjunctive query in a nonrecursive query, both with semi-interval constraints, is still in NP [30], the Π_2^P upper bound follows as in Theorem 3.2.

Example 5.1. The maximally contained query plan P_3 for query Q_3 in Example 1.1 is as follows:

$$\begin{aligned}
p_3(\mathit{CarNo},\mathit{Review})\text{:- } & \mathit{RedCars}(\mathit{CarNo},\mathit{Model},\mathit{Year}),\\
& \mathit{CarAndDriver}(\mathit{Model},\mathit{Review}),\ \mathit{Year} < 1990.\\
p_3(\mathit{CarNo},\mathit{Review})\text{:- } & \mathit{AntiqueCars}(\mathit{CarNo},\mathit{Model},\mathit{Year}),\\
& \mathit{CarAndDriver}(\mathit{Model},\mathit{Review}).
\end{aligned}$$

Because P_3 does not contain plan P_1' from Example 2.2, 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 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 datalog maximally contained query plan for Q_2 does not always exist.

First we prove a lemma similar in spirit to Lemma 4.2. It relies on the fact that, given a nonrecursive query Q_1 without comparison predicates 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, it follows from [30,33] that we can use standard algorithms for rewriting queries using views to construct P_1.

Lemma 5.1. *Given nonrecursive datalog queries Q_1 and Q_2 and conjunctive views $\mathcal{V}$, where Q_1 does not contain comparison predicates but Q_2 and $\mathcal{V}$ may contain arbitrary comparison predicates, let P_1 be the maximally contained query plan for Q_1. Then $Q_1 \sqsubseteq_{\mathcal{V}} Q_2 \Leftrightarrow P_1^{\mathrm{exp}} \sqsubseteq Q_2$.*

Proof. For the "⇒" direction,

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

For the "⇐" 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 $\bar{t}$ such that $\bar{t} \in \mathit{certain}(Q_1, I)$ but $\bar{t} \notin \mathit{certain}(Q_2, I)$. Since

$\bar{t} \notin certain(Q_2, I)$, there exists some database D such that $I \subseteq \mathcal{V}(D)$ and $\bar{t} \notin Q_2(D)$. On the other hand, since $\bar{t} \in certain(Q_1, I)$, we know that $\bar{t} \in P_1(I)$. Further, by the monotonicity of P_1, $\bar{t} \in P_1(I')$ for any I' such that $I \subseteq I'$. So in particular, $\bar{t} \in P_1(\mathcal{V}(D))$, so $\bar{t} \in P_1^{\text{exp}}(D)$. Since $\bar{t} \in P_1^{\text{exp}}(D)$ but $\bar{t} \notin Q_2(D)$, we have that $P_1^{\text{exp}} \not\sqsubseteq Q_2$. □

Using this lemma, the relative containment result follows:

Theorem 5.2. *Given nonrecursive datalog queries Q_1 and Q_2 and conjunctive views $\mathcal{V}$, where Q_1 does not contain comparison predicates but 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. By Lemma 5.1, 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 $P_1^{\text{exp}} \sqsubseteq Q_2$. Suppose Q_1 has size n. Since Q_1 is nonrecursive and does not contain comparison predicates, Lemma 3.1 still holds, so each conjunctive query plan for Q_1 need only contain at most n subgoals. Therefore, we can prove that $P_1^{\text{exp}} \sqsubseteq Q_2$, and hence that $Q_1 \sqsubseteq_{\mathcal{V}} Q_2$, by showing that for all conjunctive query plans CQ with size at most n, either $CQ^{\text{exp}} \not\sqsubseteq Q_1$ or $CQ^{\text{exp}} \sqsubseteq Q_2$.

Since Q_1 does not contain comparison predicates, checking that $CQ^{\text{exp}} \sqsubseteq Q_1$ is in NP, even though CQ^{exp} may contain comparison predicates, so checking noncontainment is in co-NP. Checking that $CQ^{\text{exp}} \sqsubseteq Q_2$ is in Π_2^P, because both queries may contain arbitrary comparison predicates [30,44]. Therefore, the algorithm for relative containment checks a property of the form

$$\forall CQ.(\forall u.\phi_1(Q_1, \mathcal{V}, CQ, u) \vee \forall v.\exists z.\phi_2(Q_2, \mathcal{V}, CQ, v, z)),$$

where CQ, u, v, and z have sizes polynomial in the sizes of Q_1, Q_2, and $\mathcal{V}$, and ϕ_1 and ϕ_2 are polynomial-time computable. By pulling all the quantifiers out front, we can see that this has the form $\forall x.\exists y.\phi(w, x, y)$, where x and y have sizes polynomial in the size of w and ϕ is computable in polynomial time. Therefore, the algorithm is in Π_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 5.2 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 conjunctive or nonrecursive 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 of 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 [1].

As the following example shows, considering sources to be complete does affect relative containment:

Example 6.1. Consider the following queries and views:

Q_1. $q_1(x,y)$:- $p(x,y)$.
Q_2. $q_2(x,y)$:- $r(x,y)$.
$v_1(x)$:- $p(x,y)$.
$v_2(y)$:- $p(x,y)$.
$v_3(x,y)$:- $p(x,y), r(x,y)$.

Under the assumption of incomplete sources, $Q_1 \sqsubseteq_{\mathscr{V}} Q_2$. In particular, views v_1 and v_2 do not provide any certain answers of Q_1, for any instance of the views. For example, if $v_1(a)$ is in a given instance of v_1, we can infer that there is some constant c such that $p(a,c)$. However, since sources are potentially incomplete, it is impossible to learn what this constant c is.

Under the assumption of complete sources, it is possible to make correlations among views to deduce information not derivable under the assumption of incomplete sources. For example, 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 $p(a,b)$ holds, so (a,b) is a certain answer of Q_1 w.r.t. I. However, Q_2 has no certain answers w.r.t. I, so $Q_1 \not\sqsubseteq_{\mathscr{V}} Q_2$.

An important subtlety under the assumption of complete sources is the fact that not all view instances are possible. In our example, the view instance $I = \{v_1(a)\}$ is not valid, because it implies the existence of a tuple of the form $p(a,c)$, but there is no witnessing instance $v_2(c)$ in I.

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 a third approach to specifying source descriptions, based on description logics [7–9].

References

[1] S. Abiteboul, O. Duschka, Complexity of answering queries using materialized views, in: Proceedings of PODS, Seattle, WA, 1998, pp. 254–263.
[2] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison–Wesley, Reading, MA, 1995.
[3] S. Adali, K. Candan, Y. Papakonstantinou, V. Subrahmanian, Query caching and optimization in distributed mediator systems, in: Proceedings of SIGMOD, Montreal, Canada, 1996, pp. 137–148.
[4] A. Aho, Y. Sagiv, J.D. Ullman, Equivalence of relational expressions, SIAM J. Comput. 2 (8) (1979) 218–246.

[5] J.L. Ambite, N. Ashish, G. Barish, C.A. Knoblock, S. Minton, P.J. Modi, I. Muslea, A. Philpot, S. Tejada, ARIADNE: A system for constructing mediators for internet sources (system demonstration), in: Proceedings of SIGMOD, Seattle, WA, 1998.
[6] C. Beeri, G. Elber, T. Milo, Y. Sagiv, O.Shmueli, N. Tishby, Y. Kogan, D. Konopnicki, P. Mogilevski, N. Slonim, Websuite—a tool suite for harnessing web data, in: Proceedings of the International Workshop on the Web and Databases, Valencia, Spain, 1998.
[7] C. Beeri, A.Y. Levy, M.-C. Rousset, Rewriting queries using views in description logics, in: Proceedings of PODS, Tucson, AZ, 1997, pp. 99–108.
[8] D. Calvanese, G.D. Giacomo, M. Lenzerini, Answering queries using views in description logics, in: Working Notes of the KRDB Workshop, Linköping, Sweden, 1999.
[9] T. Catarci, M. Lenzerini, Representing and using interschema knowledge in cooperative information systems, J. Intell. Cooperative Inform. Systems 2 (4) (1993) 375–398.
[10] A. Chandra, P. Merlin, Optimal implementation of conjunctive queries in relational databases, in: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, Boulder, Colorado, 1977, pp. 77–90.
[11] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, Optimizing queries with materialized views, in: Proceedings of ICDE, Taipei, Taiwan, 1995, pp. 190–200.
[12] S. Chaudhuri, M. Vardi, On the equivalence of recursive and nonrecursive datalog programs, in: Proceedings of PODS, San Diego, CA, 1992, pp. 55–66.
[13] C. Chen, N. Roussopoulos, Implementation and performance evaluation of the ADMS query optimizer, in: Proceedings of EDBT, Cambridge, UK, March 1994, pp. 323–336.
[14] W. Cohen, Integration of heterogeneous databases without common domains using queries based on textual similarity, in: Proceedings of SIGMOD, Seattle, WA, 1998.
[15] S. Dar, M.J. Franklin, B. Jonsson, D. Srivastava, M. Tan, Semantic data caching and replacement, in: Proceedings of VLDB, Bombay, India, 1996, pp. 330–341.
[16] D. Draper, A.Y. Halevy, D.S. Weld, The nimble xml data integration system, in: Proceedings of ICDE, Heidelberg, Germany, 2001, pp. 155–160.
[17] O. Duschka, M. Genesereth, A. Levy, Recursive query plans for data integration, J. Logic Programming (Special Issue on Logic Based Heterogeneous Information Systems) 43(1) (2000) 49–73.
[18] O.M. Duschka, M.R. Genesereth, Answering recursive queries using views, in: Proceedings of PODS, Tucson, AZ, 1997, pp. 109–116.
[19] O.M. Duschka, M.R. Genesereth, Query planning in infomaster. in: Proceedings of the ACM Symposium on Applied Computing, San Jose, CA, 1997, pp. 109–111.
[20] D. Florescu, L. Raschid, P. Valduriez, A methodology for query reformulation in cis using semantic knowledge, Internat. J. Intell. Cooperative Inform. Systems (Special issue on Formal Methods in Cooperative Information Systems) 5 (4) (1996) 431–468.
[21] M. Friedman, A. Levy, T. Millstein, Navigational plans for data integration, in: Proceedings of the National Conference on Artificial Intelligence, 1999.
[22] M. Friedman, D. Weld, Efficient execution of information gathering plans. in: Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997, pp. 785–791.
[23] M.T. Friedman, Optimization Issues in Data Integration, Ph.D. Thesis, University of Washington, 1999.
[24] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, J. Widom, The TSIMMIS project: integration of heterogeneous information sources, J. Intell. Inform. Systems 8 (2) (1997) 117–132.
[25] G. Grahne, A.O. Mendelzon, Tableau techniques for querying information sources through global schemas, in: Proceedings of ICDT, Jerusalem, Israel, 1999, pp. 332–347.
[26] A. Gupta, Y. Sagiv, J.D. Ullman, J. Widom, Constraint checking with partial information, in: Proceedings of PODS, Minneapolis, MN, 1994, 45–55.
[27] L. Haas, D. Kossmann, E. Wimmers, J. Yang, Optimizing queries across diverse data sources, in: Proceedings of VLDB, Athens, Greece, 1997.
[28] Z. Ives, D. Florescu, M. Friedman, A. Levy, D. Weld, An adaptive query execution engine for data integration, in: Proceedings of SIGMOD, 1999, pp. 299–310.

[29] A.M. Keller, J. Basu, A predicate-based caching scheme for client-server database architectures, VLDB J. 5 (1) (1996) 35–47.
[30] A. Klug, On conjunctive queries containing inequalities, J. ACM 35 (1) (1988) 146–160.
[31] C.T. Kwok, D.S. Weld, Planning to gather information, in: Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence, Portland, Oregon, 1996, pp. 32–39.
[32] E. Lambrecht, S. Kambhampati, S. Gnanaprakasam, Optimizing recursive information gathering plans, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 1999, pp. 1204–1211.
[33] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, D. Srivastava, Answering queries using views, in: Proceedings of PODS, San Jose, CA, 1995, pp. 95–104.
[34] A.Y. Levy, A. Rajaraman, J.J. Ordille, Querying heterogeneous information sources using source descriptions, in: Proceedings of VLDB, Bombay, India, 1996, pp. 251–262.
[35] A.Y. Levy, Y. Sagiv, Queries independent of updates, in: Proceedings of VLDB, Dublin, Ireland, 1993, pp. 171–181.
[36] C. Li, E. Chang, On answering queries in the presence of limited access patterns, in: Proceedings of ICDT, London, UK, 2001, pp. 219–233.
[37] T. Millstein, A. Levy, M. Friedman, Query containment for data integration systems, in: Proceedings of PODS, Dallas, TX, 2000, pp. 67–75.
[38] R. Pottinger, A. Levy, A scalable algorithm for answering queries using views, in: Proceedings of VLDB, Cairo, Egypt, 2000, pp. 484–495.
[39] A. Rajaraman, Y. Sagiv, J.D. Ullman, Answering queries using templates with binding patterns, in: Proceedings of PODS, San Jose, CA, 1995, pp. 105–112.
[40] Y. Sagiv, M. Yannakakis, Equivalence among relational expressions with the union and difference operators, J. ACM 27 (4) (1980) 633–655.
[41] O. Shmueli, Equivalence of datalog queries is undecidable, J. Logic Programming 15 (1993) 231–241.
[42] L. Stockmeyer, The polynomial-time hierarchy, Theoret. Comput. Sci. 3 (1976) 1–22.
[43] O.G. Tsatalos, M.H. Solomon, Y.E. Ioannidis, The GMAP: a versatile tool for physical data independence, VLDB J. 5 (2) (1996) 101–118.
[44] R. van der Meyden, The complexity of querying indefinite data about linearly ordered domains, in: Proceedings of PODS, San Diego, CA, 1992, pp. 331–345.
[45] H.Z. Yang, P.A. Larson, Query transformation for PSJ-queries, in: Proceedings of VLDB, Brighton, England, 1987, pp. 245–254.