Recursive Query Plans for Data Integration

Oliver M. Duschka Michael R. Genesereth

Department of Computer Science, Stanford University, Stanford, CA 94305, USA

Alon Y. Levy [1]

Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

Abstract

Generating query-answering plans for data integration systems requires to translate a user query, formulated in terms of a mediated schema, to a query that uses relations that are actually stored in data sources. Previous solutions to the translation problem produced sets of *conjunctive plans*, and were therefore limited in their ability to handle recursive queries and to exploit data sources with binding-pattern limitations and functional dependencies that are known to hold in the mediated schema. As a result, these plans were incomplete w.r.t. sources encountered in practice (i.e., produced only a subset of the possible answers). We describe the novel class of *recursive* query answering plans, which enables us to settle three open problems. First, we describe an algorithm for finding a query plan that produces the maximal set of answers from the sources for arbitrary recursive queries. Second, we extend this algorithm to use the presence of functional and full dependencies in the mediated schema. Third, we describe an algorithm for finding the maximal query plan in the presence of binding-pattern restrictions in the sources. In all three cases, recursive plans are necessary in order to obtain a maximal query plan.

1 Introduction

The problem of data integration (a.k.a. information gathering agents) has recently received considerable attention due to the growing number of structured information sources available online. The goal of data integration systems (e.g.,

[1] Work performed while the author was working in the AI Principles Research Department at AT&T Laboratories.

TSIMMIS [8,18], HERMES [2], the Internet Softbot [15], SIMS [4], the Information Manifold [23], DISCO [16,29], Occam [20], Razor [17], Infomaster [11]) is to provide a *uniform* query interface to the multiple data sources, thereby freeing the user from having to locate the relevant sources, query each one in isolation, and combine manually the information from the different sources.

Data integration systems are based on the following general architecture. The user interacts with a uniform interface in the form of a set of *global* relation names that are used in formulating queries. These relations are called the *mediated schema*. The actual data is stored in external sources, called the *source relations*. In order for the system to be able to answer queries, we must specify a *mapping* between the relations in the mediated schema and the source relations. A common method to specify these mappings (employed in [23,20,11]) is to describe each source relation as the result of a *conjunctive query* (i.e., a single Horn rule) over the relations in the mediated schema. For example, a data source containing papers authored by DB researchers would be described as follows:

$$s_1(P, A) :- paper(P),\ author(P, A),\ db(A).$$

The relations *paper*, *author* and *db* are in the mediated schema, and can be used in formulating queries, and s_1 is a source relation.

Given a query from the user, formulated in terms of the relations in the mediated schema, the system must translate it to a query that mentions *only* the source relations, because only these relations are actually available. That is, the system needs to find a query expression that mentions only the source relations, and is equivalent to the original query. The new query is called a *query plan*. The problem of finding a query plan is the same as the problem of *rewriting queries using views*. In this context, the views are the relations in the sources. The problem of rewriting queries using views has also been investigated in the database literature because of its importance for query optimization and data warehousing [32,30,6,22,26,25,10].

Most previous work has considered the problem of finding query plans where the query plan is required to be *equivalent* to the original query. In practice, the collection of available data sources may not contain *all* the information needed to answer a query, and therefore, we need to resort to *maximally-contained* plans. A maximally-contained plan provides *all* the answers that are possible to obtain from the sources, but the expression describing the plan may not be equivalent to the original query. For example, if we only have the s_1 source available, and our query asks for all papers by Computer Science researchers, then the following is a maximally-contained plan:

$$q(P) :- s_1(P, A).$$

In this article we consider several important extensions of the problem of finding a maximally-contained plan for a query using a set of data sources. In all of these extensions we show that it is not possible to find a maximally-contained plan if we restrict ourselves to non-recursive plans. Hence we introduce a new class of *recursive* query plans and show the following results:

- We describe an algorithm for finding a maximally-contained plan for cases in which the user query is recursive. We show that the problem of finding an equivalent plan in this case is undecidable.
- We describe an algorithm for finding a maximally-contained plan when functional and full dependencies are present in the mediated schema. The presence of dependencies further complicates the rewriting problem because it allows rewritings that are not valid otherwise. Furthermore, we show that in this context there does not always exist a non-recursive maximally-contained query plan.
- In practice, many data sources have limitations on the ways they can be accessed. For example, a name server of an institution, holding the addresses of its employees, will not provide the list of all employees and their addresses. Instead, it will provide the address for a *given* name. We extend our algorithms to the case in which there are limitations on sources, and they are described by the set of allowed binding patterns. In this case it is known that recursive plans may be necessary [20]. We describe an algorithm that constructs a recursive maximally-contained query plan.

Another significant advantage of our algorithms is that they are *generative*, rather than *descriptive*. Our algorithms generate the rewriting in time that is polynomial in the size of the query. In contrast, previous methods [22,26] describe the space of possible candidate rewritings, and propose heuristics for searching this space [20,23]. [2] These methods combine the process of finding a rewriting with the process of checking whether it is equivalent to the original query (which is NP-hard). In contrast, our method isolates the process of generating the maximally-contained rewriting, which can be done much more efficiently.

1.1 Related work

Previous work on this problem did not consider cases where the queries are recursive and where functional or full dependencies exist in the mediated schema.

[2] The algorithm in [23] checks whether the plans can be executed given the binding-pattern restrictions, but is not guaranteed to produce the maximally-contained rewriting when these restrictions are present. The algorithm in [20] produces only conjunctive plans that are guaranteed to adhere to the limitations on binding patterns, but is not guaranteed to compute the maximally-contained plan.

The first theoretical investigations of the problem concentrated on showing a bound on the size of the resulting query plan [22,26]. These results establish the complexity of the rewriting problem, but yield only non-deterministic algorithms for its solution. As stated above, the algorithms in [20,23] propose heuristics for searching the space of candidate plans. Huyn [19] proposed "pseudo-equivalent" rewritings in the case that no equivalent rewritings exist. These ideas were used in [25] to give an algorithm for rewriting conjunctive queries given source relations described by source descriptions.

The problem of finding query plans in the presence of binding-pattern limitations is considered in [26], but only an algorithm for finding an equivalent plan is presented. Later, Kwok and Weld [20] showed that if we restrict our plans to be sets of conjunctive queries, then there may *not be* a finite maximally-contained rewriting in the presence of binding-pattern limitations. More complex query capabilities in sources are considered in [24]. Complex capabilities are modeled by the ability of a source to answer a potentially infinite number of conjunctive queries. Hence, [24] considered how to answer queries given an infinite number of conjunctive source descriptions.

Several authors have considered the problem of rewriting queries using views for query optimization [32,6,30]. In this context, one usually requires a query plan that is equivalent to the original query. The algorithms described in [6,30] also explain how to combine the search for query plans with a traditional System-R style query optimizer. Another use of rewriting queries using views is explored in [2] for the purpose of deciding which cached answers can be used by a mediator. The algorithms described in [2] are aimed at capturing frequently occurring cases which can be detected efficiently.

1.2 Organization of the article

The article is organized as follows. Section 2 explains the basic terms we use in the discussion. Section 3 describes the construction of *inverse rules*, which is the basis for all the algorithms we describe in the article. This section also shows that the construction of the inverse rules suffices in order to compute maximally-contained query plans for recursive queries. Sections 4 and 5 describe the extensions of the algorithm in the presence of functional and full dependencies, respectively. Section 6 describes the algorithm for the case of limitations on binding patterns. The inverse rules described in Section 3 use a set of function symbols. In Section 7 we show how these function symbols can be removed, to obtain query plans that are datalog queries.

This article is the full version of two previously published conference papers [10,12]. In addition to containing the full proofs of the theorems, this

article (1) extends [12] to full dependencies and (2) shows that recursive plans are *necessary* when functional dependencies are present in the mediated schema.

2 Preliminaries

2.1 Relations and queries

We model the mediated schema and the data sources by sets of relations. For every relation, we associate an *attribute name* to each of its arguments. For example, the attribute names of the binary relation *author* may be *Paper* and *Person*. For a tuple t of a relation r with attribute A, we denote by $t[A]$ the value of the attribute A in t.

We consider datalog queries over sets of relations. A datalog query is a set of function-free Horn rules of the form

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

where p, and $p_1, \ldots, p_n$ are predicate names, and $\bar{X}$, $\bar{X}_1$, ..., $\bar{X}_n$ are tuples of variables or constants. The *head* of the rule is $p(\bar{X})$, and its *body* is $p_1(\bar{X}_1)$, ..., $p_n(\bar{X}_n)$. Each $p_i(\bar{X}_i)$ is a *subgoal* of the rule. We require that the rules be safe, i.e., every variable in the head of a rule must also occur in the body of the rule. A predicate is an *intensional database predicate*, or *IDB predicate*, in a query $\mathcal{Q}$ if it appears as the head of some rule in $\mathcal{Q}$. Predicates not appearing in any head are *extensional database predicates*, or *EDB predicates*. We assume that every query has an IDB predicate q, called the query predicate, that represents the result of $\mathcal{Q}$.

The input of a datalog query $\mathcal{Q}$ consists of a database D storing extensions of all EDB predicates in $\mathcal{Q}$. Given such a database D, a bottom-up evaluation is one in which we start with the ground EDB facts in D and apply the rules to derive facts for the IDB predicates. The output of $\mathcal{Q}$, denoted $\mathcal{Q}(D)$, is the set of ground facts generated for the query predicate in the bottom-up evaluation.

As an intermediate result of our algorithms, we will construct datalog programs with function symbols. That is, some of the arguments in the bodies or the heads of the rules are functional terms. When datalog queries contain function symbols we will refer to them as *logic queries*. In general, the bottom-up evaluation of a logic query may not terminate. As it turns out, we introduce function symbols in a controlled fashion, and in particular, the evaluation of our logic queries is guaranteed to terminate. Furthermore, we

show in Section 7 how to remove the function symbols.

Given a query, we can define a *dependency graph*, whose nodes are the predicate names appearing in the rules. There is an edge from the node of predicate p_i to the node of predicate p if p_i appears in the body of a rule whose head predicate is p. The query is *recursive* if there is a cycle in the dependency graph. A *conjunctive query* is a single non-recursive function-free Horn rule. A recursive datalog query can be seen as a finite encoding of a potentially infinite set of conjunctive queries. We do not consider interpreted predicates in this article (e.g., $\neq$, $\leq$, $<$).

2.2 Containment

A datalog query $\mathcal{Q}'$ is *contained* in a datalog query $\mathcal{Q}$ if, for all databases D, $\mathcal{Q}'(D)$ is a subset of $\mathcal{Q}(D)$. Datalog queries $\mathcal{Q}'$ and $\mathcal{Q}$ are *equivalent* if $\mathcal{Q}'$ and $\mathcal{Q}$ are contained in one another. The problem of determining whether a datalog query $\mathcal{Q}'$ is contained in a datalog query $\mathcal{Q}$ is in general undecidable [28]. The problem remains decidable if either $\mathcal{Q}'$ or $\mathcal{Q}$ are non-recursive [27,7]. In our discussion we use the following algorithm from [27] to test when a union of conjunctive queries $\mathcal{Q}'$ is contained in a recursive query $\mathcal{Q}$.[3] First, replace all variables in $\mathcal{Q}'$ by distinct constants. Consider the database D_c that contains exactly the tuples corresponding to the subgoals in the "frozen" bodies of the rules in $\mathcal{Q}'$. D_c is called the *canonical database* of $\mathcal{Q}'$. Evaluate $\mathcal{Q}$ on the canonical database. $\mathcal{Q}'$ is contained in $\mathcal{Q}$ if and only if the "frozen" heads of the rules in $\mathcal{Q}'$ are contained in $\mathcal{Q}(D_c)$.

Example 1 *Let $\mathcal{Q}$ be the following datalog query:*

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

To determine whether the non-recursive datalog query

$$\begin{aligned}\mathcal{Q}': \quad & q(X,Y) :- edge(X,Z),\ edge(Z,Y),\ black(X),\ black(Z)\\ & q(X,Y) :- edge(X,V),\ edge(V,W),\ edge(W,Y),\ black(V),\\ & \qquad\qquad black(W)\end{aligned}$$

is contained in $\mathcal{Q}$, we replace the variables in the two rules by distinct constants:

$$q(c_1,c_3) :- edge(c_1,c_2),\ edge(c_2,c_3),\ black(c_1),\ black(c_2)$$

[3] Recall that every non-recursive datalog program can be translated into an equivalent union of conjunctive queries.

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

This yields the following canonical database:

$$\underline{edge} \\ \langle c_1, c_2 \rangle, \ \langle c_2, c_3 \rangle, \ \langle c_4, c_5 \rangle, \ \langle c_5, c_6 \rangle, \ \langle c_6, c_7 \rangle \\ \underline{black} \\ \langle c_1 \rangle, \ \langle c_2 \rangle, \ \langle c_5 \rangle, \ \langle c_6 \rangle$$

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

2.3 Functional dependencies

An instance of a relation p satisfies the *functional dependency* $A_1, \ldots, A_n \rightarrow B$ if for every two tuples t and u in p with $t.A_i = u.A_i$ for $i = 1, \ldots, n$, also $t.B = u.B$. We will abbreviate a set of attributes $A_1, \ldots, A_n$ by $\bar{A}$.

When the relations satisfy a set of functional dependencies Σ, we refine our notion of containment to *relative containment*: Query $\mathcal{Q}'$ is *contained* in query $\mathcal{Q}$ *relative to* Σ, denoted $\mathcal{Q}' \subseteq_\Sigma \mathcal{Q}$, if for each database D satisfying the functional dependencies in Σ, $\mathcal{Q}'(D) \subseteq \mathcal{Q}(D)$.

In order to decide containment of conjunctive queries in the presence of functional dependencies, Aho et al. [3] show that it suffices to precede the containment algorithm by applying the *chase* algorithm to the contained query. A step in applying the chase to the body of a conjunctive query $\mathcal{Q}$ is the following. If the functional dependency $\bar{A} \rightarrow B$ holds for a relation p, and a conjunctive query $\mathcal{Q}$ has two subgoals of p, g_1 and g_2, with the same variables or values for the attributes $\bar{A}$, and g_1 has a variable X for attribute B, then we replace the occurrences of X in $\mathcal{Q}$ by the value or variable for B in g_2.

2.4 Full dependencies

Functional dependencies are a special form of a more general kind of dependencies, called full dependencies [4]. A *full dependency* δ is a first-order formula of the form

[4] Full dependencies include also two other well-known dependencies, namely multivalued dependencies and join dependencies.

$$\forall \bar{X} \; [\; \phi(\bar{X}) \Rightarrow \psi(\bar{Y}) \;]$$

where $\phi(\bar{X})$ is a conjunction of relations and equality atoms with variables $\bar{X}$, $\psi(\bar{Y})$ is a relation or an equality atom with variables $\bar{Y}$, and $\bar{Y} \subseteq \bar{X}$. If ψ is an equality atom, then δ is called an *equality generating dependency*. If ψ is a relation, then δ is called a *tuple generating dependency*. In examples, we will omit the universal quantification for the sake of brevity. A functional dependency $A \rightarrow B$ of relation $p(A, B, C)$ is an equality generating dependency because it can be written in the form

$$\forall X \forall Y \forall Z \forall Y' \forall Z' \; [\; p(X, Y, Z) \wedge p(X, Y'Z') \Rightarrow Y = Y' \;].$$

Query $\mathcal{Q}'$ is *contained* in query $\mathcal{Q}$ *relative to* a set of full dependencies Δ, denoted $\mathcal{Q}' \subseteq_\Delta \mathcal{Q}$, if for each database D satisfying the full dependencies in Δ, $\mathcal{Q}'(D) \subseteq \mathcal{Q}(D)$.

2.5 Data sources and query plans

The schema of a mediator includes a set of *virtual* relations. The relations in the mediator are virtual because their extensions are not actually stored. Their role is to provide the user a uniform interface to a multitude of data sources. We refer to the schema of the mediator as the *mediated schema*. The actual data is stored in a set of external data sources. We model each source by containing the extension of a *source relation*. The set of names of source relations is disjoint from the set of names of relations in the mediated schema.

To answer user queries, the mediator must also have a mapping between the relations in the mediated schema and source relations. We follow the approach taken in [23,20,11], where the mappings (a.k.a. source descriptions) are specified by a set of conjunctive queries, one for every source relation. The predicates in the heads of the conjunctive queries are source relations, and the predicates in their bodies are relations in the mediated schema. The meaning of such a mapping is that all the tuples that are found in the data source satisfy the query over the mediated-schema relations. [5]

Example 2 *Consider a mediated schema that includes the relations parent, male and female. The source descriptions below say that the source relations s_1 and s_2 store the father and mother relation, respectively.*

$$s_1(X, Y) :- \; parent(X, Y), \; male(X)$$

[5] Several authors have distinguished the case in which the source contains *all* the tuples that satisfy the query from the case in which some tuples may be missing from the source [13,14,9,21,1]. For our discussion this distinction does not matter.

$$s_2(X, Y) :- parent(X, Y),\ female(X)$$

Given a query $\mathcal{Q}$ from the user, the mediator needs to formulate a *query plan*, which is a query that bottoms out in the source relations and produces answers to $\mathcal{Q}$. A query plan is a set of Horn rules whose EDB predicates include *only* the source relations. The *expansion* $\mathcal{P}^{exp}$ of a query plan $\mathcal{P}$ is obtained from $\mathcal{P}$ by replacing all source relations with their corresponding source descriptions. Existentially quantified variables in source descriptions are replaced by new variables in the expansion.

Example 3 *The following query plan determines all grandparents of ann from the sources described in Example 2:*

$$\begin{aligned} q(X) &:- p(X, Z),\ p(Z, ann) \\ p(X, Y) &:- s_1(X, Y) \\ p(X, Y) &:- s_2(X, Y) \end{aligned}$$

The expansion of this query plan is the following datalog query:

$$\begin{aligned} q(X) &:- p(X, Z),\ p(Z, ann) \\ p(X, Y) &:- parent(X, Y),\ male(X) \\ p(X, Y) &:- parent(X, Y),\ female(X) \end{aligned}$$

2.6 Equivalent vs. maximally-contained query plans

A query plan $\mathcal{P}$ is contained in a datalog query $\mathcal{Q}$ if $\mathcal{P}^{exp}$ is contained in $\mathcal{Q}$, and is equivalent to $\mathcal{Q}$ if $\mathcal{P}^{exp}$ is equivalent to $\mathcal{Q}$. A query plan $\mathcal{P}$ is contained in another query plan $\mathcal{P}'$, if $\mathcal{P}^{exp}$ is contained in $(\mathcal{P}')^{exp}$. A query plan $\mathcal{P}$ is *maximally-contained* in a datalog query $\mathcal{Q}$ if $\mathcal{P}$ is contained in $\mathcal{Q}$, and for every query plan $\mathcal{P}'$ that is contained in $\mathcal{Q}$, $\mathcal{P}'$ is already contained in $\mathcal{P}$. Containment and maximal containment relative to a set of functional dependencies Σ or relative to a set of full dependencies Δ is defined accordingly. Note that the notion of maximal containment is relative to a fixed set of source relations.

Ideally, the mediator would try to find a query plan that is equivalent to the user query. However, in practice we may not have sufficient data sources to completely answer the user query. Hence, the mediator tries to find the maximally-contained query plan. In a sense, the maximally-contained query plan produces *all* the answers to the query that could be retrieved from the available sources. Of course, if there exists a plan that is equivalent to the user query then it will be a maximally-contained plan.

In this article we focus on finding maximaly-contained plans. As it turns out,

in the cases we consider in this article, the maximally-contained query plan may *have to* be a recursive datalog program. Furthermore, we show that if the query $\mathcal{Q}$ is recursive, then finding an equivalent query plan is undecidable, while finding a maximally-contained query plan is decidable.

3 Inverse rules and recursive queries

In this section we first describe how to compute a set of *inverse rules* from a given set of source descriptions. Intuitively, inverse rules can be viewed as query plans for the predicates in the mediated schema. Inverse rules are common to all the constructions we describe in this article. We then show that the inverse rules themselves, together with a recursive datalog query $\mathcal{Q}$ provide a maximally-contained plan for $\mathcal{Q}$. It should be noted that previous work considered the construction of query plans only for non-recursive datalog queries. Finally, we show that the problem of finding an equivalent query plan for recursive queries is undecidable.

As explained below, in constructing the inverse rules we use function symbols. These function symbols can later be eliminated, as we will show in Section 7. We use the following set of function symbols in inverse rules. For every source relation s with variables $X_1, \ldots, X_n$ in the body but not in the head of its source description, we have a function symbol $f_{s,i}$. The arity of the function $f_{s,i}$ is the arity of s.

Definition 4 (inverse rules) *Let s be a source relation defined by the source description*

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

Then for $j = 1, \ldots, n$,

$$p_j(\bar{X}'_j) :- s(\bar{X})$$

is an inverse rule *of s, denoted s^{-1}. We modify $\bar{X}_j$ to obtain the tuple $\bar{X}'_j$ as follows: if X is a constant or is a variable in $\bar{X}$, then X is unchanged in X'_j. Otherwise, X is one of the variables X_i appearing in the body of s but not in $\bar{X}$, and X is replaced by $f_{s,i}(\bar{X})$ in $\bar{X}'_j$.*

We denote the set of inverse rules of the source descriptions in $\mathcal{V}$ by $\mathcal{V}^{-1}$.

Example 5 *The inverse of the source descriptions*

$$\begin{aligned} s_1(X,Y) &:- edge(X,Z),\ edge(Z,W),\ edge(W,Y) \\ s_2(X) &:- edge(X,Z) \end{aligned}$$

is the following set of rules (to simplify notation, we use f_1 for $f_{s_1,1}$, f_2 for $f_{s_1,2}$, and f_3 for $f_{s_2,1}$):

$$\begin{array}{ll} edge(X, f_1(X,Y)) & :- \ s_1(X,Y) \\ edge(f_1(X,Y), f_2(X,Y)) & :- \ s_1(X,Y) \\ edge(f_2(X,Y), Y) & :- \ s_1(X,Y) \\ edge(X, f_3(X)) & :- \ s_2(X) \end{array}$$

Given a datalog query $\mathcal{Q}$ and a set of conjunctive source descriptions $\mathcal{V}$, the construction of the query plan is as follows. We delete all rules from $\mathcal{Q}$ that contain mediated schema relations that do not appear in any of the source descriptions. To the resulting query, denoted as $\mathcal{Q}^-$, we add the rules of $\mathcal{V}^{-1}$, and call the query so obtained $(\mathcal{Q}^-, \mathcal{V}^{-1})$. Notice that the EDB predicates of the remaining rules of $\mathcal{Q}$ are IDB predicates in $(\mathcal{Q}^-, \mathcal{V}^{-1})$, because they appear in heads of the rules in $\mathcal{V}^{-1}$. Because naming of IDB predicates is arbitrary, one could rename the IDB predicates in $(\mathcal{Q}^-, \mathcal{V}^{-1})$ so that their names differ from the names of the corresponding EDB predicates in $\mathcal{Q}$. For ease of exposition, we will not do it here.

Example 6 *Consider the recursive query*

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

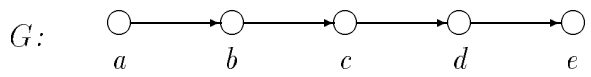
which determines the transitive closure of the relation edge. Assume there is only one data source available:

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

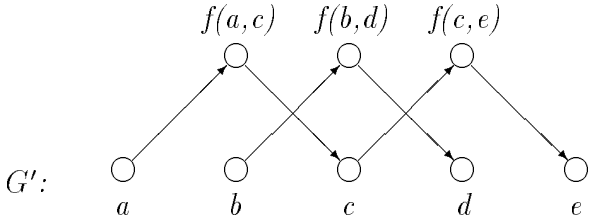
Source s stores endpoints of paths of length two. Just using this source, there is no way to determine the transitive closure of the relation edge. The best one can hope to achieve is to compute the endpoints of paths of even *lengths. Relation edge, the only EDB predicate in $\mathcal{Q}$, appears in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

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

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



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



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

The query $(\mathcal{Q}^-, \mathcal{V}^{-1})$ is a logic query because the inverse rules contain function symbols. In order to show that it is the maximally-contained plan of $\mathcal{Q}$, we first show that the evaluation of $(\mathcal{Q}^-, \mathcal{V}^{-1})$ will terminate on every database. The key observation is that function symbols are *only* introduced in inverse rules. Because inverse rules are not recursive, no terms with nested function symbols can be generated.

Lemma 7 *For every datalog query $\mathcal{Q}$, every set of conjunctive source descriptions $\mathcal{V}$, and all finite instances of the source relations, the logic query $(\mathcal{Q}^-, \mathcal{V}^{-1})$ has a unique finite minimal fixpoint. Furthermore, bottom-up evaluation is guaranteed to terminate, and produces this unique fixpoint.*

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

Given extensions for its EDB predicates, a logic query might produce tuples containing function symbols in its result. Because the extensions of EDB predicates do not contain any function symbols, no datalog query produces tuples in its result containing function symbols. Hence, in order to compare between the result of evaluating $\mathcal{Q}$ to that of evaluating $(\mathcal{Q}^-, \mathcal{V}^{-1})$ on a set of data

sources, we need to define a filter that gets rid of all extraneous tuples with functional terms. If $\mathcal{D}$ is a set of sources containing tuples of the EDB predicates of a query plan with function symbols $\mathcal{P}$, then let $\mathcal{P}(\mathcal{D})\downarrow$ be the set of all tuples in $\mathcal{P}(\mathcal{D})$ that do not contain function symbols. Let $\mathcal{P}\downarrow$ be the plan that given the sources $\mathcal{D}$ computes $\mathcal{P}(\mathcal{D})\downarrow$.

The following theorem shows that the simple construction of adding the inverse rules to $\mathcal{Q}^-$ yields a logic query that uses the source relations in the best possible way. That is, after discarding all tuples containing function symbols, the result of $(\mathcal{Q}^-, \mathcal{V}^{-1})$ is contained in $\mathcal{Q}$. Moreover, the result of every query plan that is contained in $\mathcal{Q}$ is already contained in $(\mathcal{Q}^-, \mathcal{V}^{-1})$.

Theorem 8 *For every datalog query $\mathcal{Q}$ and every set of conjunctive source descriptions $\mathcal{V}$, the query plan $(\mathcal{Q}^-, \mathcal{V}^{-1})\downarrow$ is maximally-contained in $\mathcal{Q}$. Moreover, $(\mathcal{Q}^-, \mathcal{V}^{-1})$ can be constructed in time polynomial in the size of $\mathcal{Q}$ and $\mathcal{V}$.*

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

Let $\mathcal{P}$ be an arbitrary query plan contained in $\mathcal{Q}$. We have to prove that $\mathcal{P}$ is also contained in $(\mathcal{Q}^-, \mathcal{V}^{-1})$. Let c_s be an arbitrary conjunctive query generated by $\mathcal{P}$. If we can prove that c_s^{exp} is contained in $(\mathcal{Q}^-, \mathcal{V}^{-1})$, then $\mathcal{P}$ is contained in $(\mathcal{Q}^-, \mathcal{V}^{-1})$, which proves the claim. Let D_c be the canonical database of c_s^{exp}. Because c_s^{exp} is contained in $\mathcal{Q}$, $c_s^{exp}(D_c)$ is contained in the output of $\mathcal{Q}$ applied to D_c. Let c be the conjunctive query generated by $\mathcal{Q}$ that produces $c_s^{exp}(D_c)$. Because all predicates of query c are also in c_s^{exp}, and all predicates in c_s^{exp} appear in some source description, c is also generated by $\mathcal{Q}^-$. Because c_s^{exp} is contained in c, there is a containment mapping h from c to c_s^{exp} [5]. Every variable Z in c_s^{exp} that does not appear in c_s is existentially quantified in some source description $s_i(X_1, \ldots, X_m)$ in c_s. Let k be the mapping that maps every such variable Z to the corresponding term $f_{s_i,j}(X_1, \ldots, X_m)$ used in s_i^{-1}. Because $\mathcal{Q}^-$ can derive c, $\mathcal{Q}^-$ can also derive the more specialized conjunctive query $k(h(c))$. Using rules in $\mathcal{V}^{-1}$, the derivation of $k(h(c))$ in $\mathcal{Q}^-$ can be extended to a derivation of a conjunctive query c' that only contains

source relations. The identity mapping is a containment mapping from c' to c_s. This proves that $\mathcal{P}$ is contained in $(\mathcal{Q}^-, \mathcal{V}^{-1})$.

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

As stated earlier, if there exists an equivalent plan for a query $\mathcal{Q}$, it will be a maximally-contained plan. However, since equivalence of datalog programs is undecidable in general, we cannot test whether $(\mathcal{Q}^-, \mathcal{V}^{-1})$ is an equivalent plan by testing whether it is equivalent to $\mathcal{Q}$. Moreover, the following theorem shows that the problem of whether there exists a query plan equivalent to $\mathcal{Q}$ is undecidable.

Theorem 9 *Given a datalog query $\mathcal{Q}$ and conjunctive source descriptions, it is undecidable whether there is a query plan $\mathcal{P}$ equivalent to $\mathcal{Q}$.*

PROOF. Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two arbitrary datalog queries. We show that a decision procedure for the above problem would allow us to decide whether $\mathcal{Q}_1$ is contained in $\mathcal{Q}_2$. Because the containment problem for datalog queries is undecidable [28], this proves the claim. Without loss of generality we can assume that there are no IDB predicates with the same name in $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and that the query predicates in $\mathcal{Q}_1$ and $\mathcal{Q}_2$, named q_1 and q_2 respectively, have arity m. Let $\mathcal{Q}$ be the datalog query consisting of all the rules in $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and of the rules

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

where e is a new zero-ary global relation. Furthermore, for every global relation $e_i(X_1, \ldots, X_{k_i})$ in $\mathcal{Q}_1$ and $\mathcal{Q}_2$ (but not for e) assume there is a source relation described by the source description

$$s_i(X_1, \ldots, X_{k_i}) :- e_i(X_1 \ldots, X_{k_i}).$$

We show that $\mathcal{Q}_1$ is contained in $\mathcal{Q}_2$ if and only if there is a query plan $\mathcal{P}$ equivalent to $\mathcal{Q}$.

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

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

"$\Leftarrow$": Assume there is a query plan $\mathcal{P}$ equivalent to $\mathcal{Q}$. Then for any instantiation of the global relations, $\mathcal{Q}$ and $\mathcal{P}^{exp}$ yield the same result, especially for instantiations where e is the empty relation, and where e contains the empty tuple. If e is the empty relation then $\mathcal{Q}$ produces exactly the tuples produced by $\mathcal{Q}_2$, and therefore $\mathcal{P}^{exp}$ does likewise. If e contains the empty tuple then $\mathcal{Q}$ produces the union of the tuples produced by $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and hence $\mathcal{P}^{exp}$ produces this union. Because $\mathcal{P}^{exp}$ does not contain e, $\mathcal{P}^{exp}$ will produce the same set of tuples regardless of the instantiation of e. It follows that $\mathcal{Q}_2$ is equivalent to the union of $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Therefore, $\mathcal{Q}_1$ is contained in $\mathcal{Q}_2$. $\square$

4 Mediated schemata with functional dependencies

In this section we consider the problem of generating a maximally-contained plan for a query $\mathcal{Q}$ in the presence of functional dependencies in the mediated schema. We begin by describing an algorithm for generating a maximally-contained plan, and in the end of the section we show recursive plans may be *necessary* in this context. That is, if we restrict ourselves to plans that are unions of conjunctive queries, then we may not obtain all the possible answers from the data sources.

We use the following example throughout this section to illustrate the difficulties introduced by functional dependencies and to present our algorithm. Suppose we have the following relations in the mediated schema

$$\begin{array}{l} conference(Paper, Conference), \\ year(Paper, Year), \\ location(Conference, Year, Location) \end{array}$$

The relations describe the conference at which a paper was presented, the publication year of a paper, and the location a conference was held at in a given year. A paper is only presented at one conference and published in one year. Also, in a given year a conference is held at a specific location. Therefore we have three functional dependencies:

$$\begin{array}{ll} conference\text{:} & Paper \rightarrow Conference \\ year\text{:} & Paper \rightarrow Year \\ location\text{:} & Conference,\ Year \rightarrow Location \end{array}$$

Suppose we have the following data sources:

$$\begin{array}{ll} s_1(P, C, Y) & :- conference(P, C),\ year(P, Y) \\ s_2(P, L) & :- conference(P, C),\ year(P, Y),\ location(C, Y, L) \end{array}$$

s_1 tells us in which conference and year a paper was presented, and s_2 stores the location of the presentation of a paper directly with the paper. Assume a user wants to know where PODS '89 was held:

$$q(L) :- \mathit{location}(\mathit{pods}, 1989, L)$$

The following plan would answer the query:

$$q(L) :- s_1(P, \mathit{pods}, 1989),\ s_2(P, L)$$

Informally, the query plan proceeds as follows. It first finds *some* paper presented at PODS '89 using s_1, and then finds the location of the conference this paper was presented at using s_2. This plan is correct *only* because every paper is presented at one conference and in one year. In fact, if these dependencies would not hold, there would be no way of answering this query using the given sources. It is also important to note that source relation s_1 is needed in the query plan even though the predicates in s_1, *conference* and *year*, don't appear in the query at all. Without functional dependencies, only source descriptions that contain predicates appearing in the user query need to be considered [22].

In the following we are going to give a construction of query plans that is guaranteed to be maximally-contained in the given queries, even in the presence of functional dependencies. As in the previous section, we begin by computing the set of inverse rules, whose purpose is to recover tuples of the mediated-schema relations from the source relations. The inverse rules for s_1 and s_2 in our example are:

$$\begin{aligned}
r_1: &\quad \mathit{conference}(P, C) &&:- s_1(P, C, Y)\\
r_2: &\quad \mathit{year}(P, Y) &&:- s_1(P, C, Y)\\
r_3: &\quad \mathit{conference}(P, f_1(P, L)) &&:- s_2(P, L)\\
r_4: &\quad \mathit{year}(P, f_2(P, L)) &&:- s_2(P, L)\\
r_5: &\quad \mathit{location}(f_1(P, L), f_2(P, L), L) &&:- s_2(P, L)
\end{aligned}$$

For example, rule r_5 extracts from s_2 that *some* conference in *some* year was held in location L. Suppose that s_1 stores the information that the paper "Bottom-Up Beats Top-Down for Datalog" (henceforth abbreviated as *datalog*) was presented at PODS '89, and s_2 stores the information that "Bottom-Up Beats Top-Down for Datalog" was presented in Philadelphia. The inverse rules derive the following fats:

$$\begin{array}{ll}
\underline{\mathit{conference}} & \\
\langle \mathit{datalog},\ \mathit{pods} \rangle & (r_1)\\
\langle \mathit{datalog},\ f_1(\mathit{datalog}, \mathit{philadelphia}) \rangle & (r_3)
\end{array}$$

$$\begin{array}{ll}
\underline{\mathit{year}} & \\
\langle \mathit{datalog},\ 1989 \rangle & (r_2)
\end{array}$$

$\langle datalog,\ f_2(datalog, philadelphia)\rangle$ (r_4)

location

$\langle f_1(datalog, philadelphia),\ f_2(datalog, philadelphia),\ philadelphia\rangle$ (r_5)

The inverse rules don't take into account the presence of the functional dependencies. For example, because of the functional dependency in relation *conference*, $Paper \rightarrow Conference$, it is possible to conclude that the function term $f_1(datalog, philadelphia)$ must actually be the same as the constant *pods*. We model this inference by introducing a new binary relation e. The intended meaning of e is that $e(c_1, c_2)$ holds if and only if c_1 and c_2 must be equal under the given functional dependencies. Hence, the extension of e includes the extension of $=$ (i.e., for every X, $e(X, X)$), and the tuples that can be derived by the following chase rules ($e(\bar{A}, \bar{A}')$ is a shorthand for $e(A_1, A_1'), \ldots, e(A_n, A_n')$):[6]

Definition 10 (chase rules) *Let $\bar{A} \rightarrow B$ be a functional dependency satisfied by a relation p in the mediated schema. Let $\bar{C}$ be the attributes of p that are not in $\bar{A}, B$. The* chase rule *corresponding to $\bar{A} \rightarrow B$, denoted $chase(\bar{A} \rightarrow B)$, is the following rule:*

$$e(B, B') :- p(\bar{A}, B, \bar{C}),\ p(\bar{A}', B', \bar{C}'),\ e(\bar{A}, \bar{A}').$$

We denote by $chase(\Sigma)$ the set of chase rules corresponding to the functional dependencies in Σ. In our example, the chase rules are

$$\begin{aligned}
e(C, C') &:- conference(P, C),\ conference(P', C'),\ e(P, P') \\
e(Y, Y') &:- year(P, Y),\ year(P', Y'),\ e(P, P') \\
e(L, L') &:- location(C, Y, L),\ location(C', Y', L'),\ e(C'C'), \\
&\qquad e(Y, Y')
\end{aligned}$$

The chase rules allow us to derive the following facts in relation e:

e

$\langle f_1(datalog, philadelphia),\ pods\rangle$

$\langle f_2(datalog, philadelphia),\ 1989\rangle$

The extension of e is reflexive by construction, and is symmetric because of the symmetry in the chase rules. To guarantee that e is an equivalence relation, it is still needed to enforce transitivity of e. The following rule, denoted by $\mathcal{T}$, is sufficient for guaranteeing transitivity of relation e:

[6] We only describe relation e to be reflexive for ease of exposition. For every rule r having a subgoal $e(X, Y)$ in its body, we could add a modified version of rule r with subgoal $e(X, Y)$ removed and X replaced by Y. The resulting set of rules wouldn't require e to be reflexive.

$$e(X, Y) :- e(X, Z),\ e(Z, Y).$$

The final step in the construction is to rewrite query $\mathcal{Q}$ in a way that it can use the equivalences derived in relation e. We define the *rectified* query $\bar{\mathcal{Q}}$ by modifying $\mathcal{Q}$ iteratively as follows:

(i) If c is a constant in one of the subgoals of $\mathcal{Q}$, we replace it by a new variable Z, and add the subgoal $e(Z, c)$.
(ii) If X is a variable in the head of $\mathcal{Q}$, we replace X in the body of $\mathcal{Q}$ by a new variable X', and add the subgoal $e(X', X)$.
(iii) If a variable Y that is not in the head of $\mathcal{Q}$ appears in two subgoals of $\mathcal{Q}$, we replace one of its occurrences by Y', and add the subgoal $e(Y', Y)$.

We apply the above steps until no additional changes can be made to the query. In our example query $\mathcal{Q}$ would be rewritten to

$$\bar{q}(L) :- location(C, Y, L'),\ e(C, pods),\ e(Y, 1989),\ e(L', L),$$

Note that evaluating query $\bar{q}$ on the reconstructed relations of the mediated schema and the derived equivalence relation e yields the desired result: PODS '89 was held in Philadelphia.

Given a query q, a set of source descriptions $\mathcal{V}$, and a set of functional dependencies Σ, the constructed query plan includes $\bar{\mathcal{Q}}$, the inverse rules $\mathcal{V}^{-1}$, the chase rules $chase(\Sigma)$ and the transitivity rule $\mathcal{T}$. The following theorem shows that this query plan is maximally-contained in $\mathcal{Q}$ relative to Σ.

Theorem 11 *Let Σ be a set of functional dependencies, $\mathcal{V}$ a set of conjunctive source descriptions, and let $\mathcal{Q}$ be a conjunctive query over the relations in the mediated schema. Let $\mathcal{R}$ denote the set of rules $\mathcal{V}^{-1} \cup chase(\Sigma) \cup \mathcal{T}$. Then, $(\bar{\mathcal{Q}}, \mathcal{R})\downarrow$ is maximally-contained in $\mathcal{Q}$ relative to Σ. Furthermore, $(\bar{\mathcal{Q}}, \mathcal{R})$ can be constructed in time polynomial in the size of $\mathcal{Q}$, $\mathcal{V}$, and Σ.* □

PROOF. The key to the proof is to show that for every conjunctive query plan $\mathcal{P} \subseteq_\Sigma \mathcal{Q}$, $\mathcal{P}^{exp} \subseteq_\Sigma (\bar{\mathcal{Q}}, \mathcal{R})^{exp}$. Because recursive query plans can be seen as an encoding of the union of infinitely many conjunctive query plans, it suffices to prove the claim for all conjunctive query plans. We prove the following statement by induction on k: if $\mathcal{Q}$ is a query, $\mathcal{P}$ is a conjunctive query plan, and $e_1, \ldots, e_k$ is a sequence of queries with $e_1 = \mathcal{P}^{exp}$, $e_k \subseteq \mathcal{Q}$, and e_{i+1} results from e_i by applying a chase step, then $\mathcal{P}^{exp} \subseteq_\Sigma (\bar{\mathcal{Q}}, \mathcal{R})^{exp}$. This would prove that $(\bar{\mathcal{Q}}, \mathcal{R})$ is maximally-contained in $\mathcal{Q}$ relative to Σ.

For $k = 1$, $\mathcal{P}^{exp}$ is contained in $\mathcal{Q}$. As shown in Theorem 8, this implies that $\mathcal{P}^{exp}$ is contained in $(\mathcal{Q}, \mathcal{V}^{-1})^{exp}$. It follows that $\mathcal{P}^{exp}$ is contained in $(\bar{\mathcal{Q}}, \mathcal{R})^{exp}$ relative to Σ.

For the induction step, let $k > 1$ and assume $e_{k-1} \not\subseteq q$. Let $\bar{A} \rightarrow B$ be the functional dependency that holds for relation p and that is applied from e_{k-1} to e_k. Then e_{k-1} contains two subgoals of p, g_1 and g_2, with the same values/variables for the attributes in $\bar{A}$, and g_1 contains a variable X for attribute B that is replaced by some value/variable in e_k. Let h be the containment mapping [5] that shows that $\mathcal{Q}$ contains e_k. Replace every value/variable X_i in an argument position in $\mathcal{Q}$ that is mapped by h to an argument position in e_k that used to be variable X in e_{k-1} by a new variable X_i'. For each of the new variables X_i', add two subgoals of p to $\mathcal{Q}$ with the identical new variables for the corresponding attributes $\bar{A}$, X_i and X_i' for attribute B respectively, and new variables for the remaining attributes. We can now find a containment mapping from query $\mathcal{Q}'$ to query e_{k-1}. This shows that e_{k-1} is contained in q'. Therefore, $\mathcal{P}^{exp} \equiv e_1, \ldots, e_{k-1}$ is a chase sequence with $e_{k-1} \subseteq \mathcal{Q}'$. By the induction hypothesis we have that $\mathcal{P}^{exp} \subseteq (\bar{\mathcal{Q}}', \mathcal{R})^{exp}$. Using the chase rule $chase(\bar{A} \rightarrow B)$, the transitivity rule, and the reflexivity of relation e, we can show that $(\bar{\mathcal{Q}}', \mathcal{R}) \subseteq (\bar{\mathcal{Q}}, \mathcal{R})$. It follows that $\mathcal{P}^{exp} \subseteq_\Sigma (\bar{\mathcal{Q}}, \mathcal{R})^{exp}$.

Query $\bar{\mathcal{Q}}$ contains all subgoals in q, and at most as many additional subgoals of e as the sum of all arities of the subgoals in $\mathcal{Q}$. Also, there are as many inverse rules as there are subgoals in all source descriptions in $\mathcal{V}$ together. Finally, there are exactly as many chase rules as there are functional dependencies in Σ. We can conclude that $(\bar{\mathcal{Q}}, \mathcal{R})$ can be constructed in time polynomial in the size of $\mathcal{Q}$, $\mathcal{V}$ and Σ. □

We showed that recursive query plans are expressive enough to extract the maximal amount of information from the data sources even in the presence of functional dependencies. Still, one might ask whether it is somehow possible to do without recursion in the plans. The following example shows that recursion is really needed in order not to miss any answers.

Example 12 *Suppose we have the following relation in the mediated schema*

$$schedule(Airline, Flight\_no, Date, Pilot, Aircraft)$$

which represents the pilot that is scheduled for a certain flight, and the aircraft that is used for this flight. Assume we have the following functional dependencies on the relations in the mediated schema

$$Pilot \rightarrow Airline \quad and$$
$$Aircraft \rightarrow Airline$$

expressing that pilots work for only one airline, and that there is no joint ownership of aircraft between airlines. The following data source is available:

$$s_3(D, P, C) :- schedule(A, N, D, P, C)$$

s_3 records on which date which pilot flies which aircraft. Assume a user asks for pilots that work for the same airline as Mike:

$$q(P) :- schedule(A, N, D, mike, C),\ schedule(A, N', D', P, C')$$

Source s_3 doesn't record the airlines that pilots work for. Nonetheless, using the functional dependencies of relation schedule, conclusions can be drawn on which pilots work for the same airline as Mike. For example, if both Mike and Ann are known to have flown aircraft #111, then Ann works for the same airline as Mike because of the functional dependency Aircraft $\rightarrow$ Airline. Moreover, if Ann is known to have flown aircraft #222, and John has flown aircraft #222 then Ann and John work for the same airline because of the second functional dependency. Hence, we can infer that John and Mike work for the same airline. In general, the query plan $\mathcal{P}_n$ given by

$$\begin{aligned} q_n(P) :- & \ s_3(D_1, mike, C_1),\ s_3(D_2, P_2, C_1),\ s_3(D_3, P_2, C_2), \\ & \ s_3(D_4, P_3, C_2),\ \ldots,\ s_3(D_{2n-2}, P_n, C_{n-1}), \\ & \ s_3(D_{2n-1}, P_n, C_n),\ s_3(D_{2n}, P, C_n) \end{aligned}$$

is contained in the user query for each n. Moreover, each $\mathcal{P}_n$ is not contained in any shorter query plan. This means that any non-recursive query plan with a fixed number of subgoals cannot be maximally-contained in the user query.

5 Full dependencies

In this section we generalize the algorithm of the previous section to arbitrary full dependencies. The added expressive power of full dependencies allows, for example, to express constraints between different relations. As an example, assume that United Airlines as a rule always uses one specific aircraft for every connection, in both directions. This can be expressed by the following full dependencies:

$$\begin{aligned} & schedule(ua, N, D, P, C) \wedge schedule(ua, N, D', P', C') \ \Rightarrow\ C = C' \\ & flight(ua, N, F, T) \wedge flight(ua, N', T, F) \wedge \\ & \quad schedule(ua, N, D, P, C) \wedge schedule(ua, N', D', P', C') \ \Rightarrow\ C = C' \end{aligned}$$

The first full dependency expresses that United Airlines operates only one aircraft for every flight number. The second full dependency states that the aircraft used in both directions are the same.

The key to generalizing our algorithm is to define chase rules for these more general dependencies. Let δ be a full dependency. The rectified full dependency $\overline{\delta}$ can be obtained from δ by rectifying the antecedent of its implication using

the same procedure as for rectifying queries presented in Section 4. For example, the rectified version of the first full dependency above is the following full dependency:

$$schedule(A, N, D, P, C) \wedge schedule(A', N', D', P', C') \wedge$$
$$A = ua \wedge A' = ua \wedge N = N' \;\Rightarrow\; C = C'$$

For every full dependency there is an equivalent rectified full dependency. Therefore, it suffices to define generalized chase rules for rectified full dependencies only.

Definition 13 (generalized chase rules) *Let*

$$\forall \bar{X}\ [\ p_1(\bar{X}_1) \wedge \ldots \wedge p_{n-1}(\bar{X}_{n-1}) \;\Rightarrow\; p_n(\bar{X}_n)\ ]$$

be a rectified full dependency, where $p_1, \ldots, p_n$ are either mediated-schema relations or equality atoms. The generalized chase rule *corresponding to this full dependency is the following rule:*

$$\bar{p}_n(\bar{X}_n) :- \bar{p}_1(\bar{X}_1),\ \ldots,\ \bar{p}_{n-1}(\bar{X}_{n-1}).$$

If p_i is a mediated-schema relation, then $\bar{p}_i$ is p_i. Otherwise, p_i is an equality atom $Y_i = Z_i$, and then $\bar{p}_i$ is defined to be $e(Y_i, Z_i)$.

We denote by $chase(\Delta)$ the set of generalized chase rules corresponding to the full dependencies in Δ. The generalized chase rule corresponding to the rectified full dependency mentioned above is the following rule:

$$e(C, C') :- schedule(A, N, D, P, C),\ schedule(A', N', D', P', C'),$$
$$e(A, ua),\ e(A', ua),\ e(N, N')$$

Note that for functional dependencies, generalized chase rules are identical to the corresponding chase rules defined in Section 4. To generate a maximally-contained plan in the presence of full dependencies, we follow the same algorithm as in Section 4, except that we replace the chase rules by the generalized chase rules. The following theorem generalizes Theorem 11.

Theorem 14 *Let Δ be a set of full dependencies, $\mathcal{V}$ a set of conjunctive source descriptions, and let $\mathcal{Q}$ be a query over the relations in the mediated schema. Let $\mathcal{R}$ denote the set of rules $\mathcal{V}^{-1} \cup chase(\Delta) \cup \mathcal{T}$. Then, $(\bar{\mathcal{Q}}, \mathcal{R})\downarrow$ is maximally-contained in $\mathcal{Q}$ relative to Δ. Furthermore, $(\bar{\mathcal{Q}}, \mathcal{R})$ can be constructed in time polynomial in the size of $\mathcal{Q}$, $\mathcal{V}$, and Δ.* □

The proof of the theorem is similar to that of Theorem 11, noting that in the presence of full dependencies, it also suffices to precede a containment test by the chase algorithm.

The dependencies that we consider in this chapter are called *full* dependencies because all the variables that appear on the right hand side of the implication in a dependency must already occur on the left hand side. This restrictions is essential. The following dependency is *not* a full dependency:

$$\mathit{flight}(A, N, F, T) \Rightarrow \exists N' \, \mathit{flight}(A, N', T, F)$$

This kind of dependency is usually refered to as an *inclusion dependency* because it asserts that the set of values appearing for some attribute is included in the set of values appearing for some other attribute. The dependency expresses that if an airline offers a flight between two cities, then the airline offers the flight in both directions. If we allowed this kind of dependency, the corresponding chase rule would be

$$\mathit{flight}(A, f(A, N, F, T), T, F) :- \mathit{flight}(A, N, F, T).$$

But this rule is recursive and introduces new function terms. Therefore, naive bottom-up evaluation of a query containing this rule wouldn't terminate. The question of whether it is possible to build the maximally-contained query plan in the presence of general — including non-full — dependencies remains open.

6 Limitations on binding patterns

The last case we consider in this article is the presence of limitations on access to data sources. In practice, some information sources cannot answer arbitrary atomic queries on the relation they store. In particular, the data source may require that some of the arguments of its relations be given as input. To model source capabilities, we attach to each source relation an *adornment* (see [31], Chap. 12), specifying which binding patterns the source supports.[7] An adornment of a source description of s is a string of b's and f's of length n, where n is the arity of s. The meaning of the adornment is that the source only supports queries in which the arguments with b adornments are bound. The other arguments may be either bound or free. For example, the adornment s^{bf} means that the first argument must be bound in queries on s. We define an *executable* query plan as follows.

Definition 15 (executable query plan) *Let $\mathcal{V}$ be a set of source descriptions with binding adornments, and let $\mathcal{P}$ be the following conjunctive query plan:*

$$q(\bar{X}) :- s_1(\bar{X}_1), \ldots, s_n(\bar{X}_n)$$

[7] For simplicity of exposition, we assume that each source relation has a single adornment.

Query plan $\mathcal{P}$ is executable *if the following holds for $i = 1, \ldots, n$: let j be an argument position of s_i that has a b adornment, and let α be the j'th element in $\bar{X}_i$. Then, either α is a constant, or α appears in $\bar{X}_1 \cup \ldots \cup \bar{X}_{i-1}$.*

A datalog query plan includes source relations and IDB relations. We model the IDB relations as having the all-free adornment (i.e., f^n, where n is the relation's arity). A query plan $\mathcal{P}$ is executable if for every rule $r \in \mathcal{P}$, r is executable.

In [26] it is shown that in the existence of binding pattern limitations, if we are looking for a query plan that is equivalent to the user query, then there is a bound on the number of literals we need to consider in candidate query plans. However, as the following example, adapted from [20] shows, there may *not* be a finite maximally-contained query plan, if we restrict ourselves to query plans without recursion.

Example 16 *Consider the following sources:*

$$\begin{aligned} s_1^f(X) &:- podsPapers(X) \\ s_2^{bf}(X,Y) &:- cites(X,Y) \\ s_3^b(X) &:- awardPaper(X) \end{aligned}$$

The first source stores PODS papers, the second is a citation database, but only accepts queries where the first argument is bound, and the third source will tell us whether a given *paper won an award. Suppose our query is to find all the award papers:*

$$q(X) :- awardPaper(X)$$

For each n, the following is an executable conjunctive query plan $\mathcal{P}_n$ that is contained in $\mathcal{Q}$:

$$q_n(Z_n) :- s_1(Z_0),\ s_2(Z_0, Z_1),\ \ldots,\ s_2(Z_{n-1}, Z_n),\ s_3(Z_n).$$

Furthermore, for each n, $\mathcal{P}_n$ may produce answers that are not obtained by any other $\mathcal{P}_i$, for any i. Intuitively, a paper will be in the answer to $\mathcal{P}_i$ if the number of links that need to be followed from a PODS paper is i. Therefore, there is no bound on the size of the conjunctive queries in the maximally-contained plan.

We now show that by allowing recursive plans we can *produce a maximally-contained query plan. In our example, the construction will include a new recursively-defined relation, papers, whose extension will be the set of all papers that can be reached from the papers in s_1. The construction will result in the following plan.*

$$papers(X) :- s_1^f(X)$$

$$papers(X) :- papers(Y),\ s_2^{bf}(Y, X)$$
$$q(X) :- papers(X),\ s_3^{b}(X).$$

We now describe the construction for a given set of adorned source relations $\mathcal{V}$ and a query $\mathcal{Q}$. The recursive plan includes a unary relation *dom* whose intended extension is the set of all constants that appear in the query or in the source descriptions, or that can be obtained by iteratively querying the sources. The rules involving *dom* are the following:

Definition 17 (domain rules) *Let $s \in \mathcal{V}$ be a source relation of arity n. Suppose the adornment of s says that the arguments in positions $1, \ldots, l$ need to be bound, and the arguments $l+1, \ldots, n$ can be free. Then for $i = l+1, \ldots, n$, the following rule is a* domain rule*:*

$$dom(X_i) :- dom(X_1),\ \ldots,\ dom(X_l),\ s(X_1, \ldots, X_n).$$

Also, if c is a constant appearing in the source descriptions in $\mathcal{V}$ or in query q, then the fact $dom(c)$ is a domain rule*.*

We denote by $domain(\mathcal{V}, \mathcal{Q})$ the set of rules described above for defining the predicate *dom*. Notice that all domain rules are executable, and that relation *dom* has adornment f. Every query plan $\mathcal{P}$ can be transformed to an executable query plan by inserting the literal $dom(X)$ before subgoals g in $\mathcal{P}$ that have a variable X in an argument position that is required to be bound, and X does not appear in the subgoals to the left of g in the body. The resulting query plan, denoted by $\mathcal{P}^{exec}$, is executable. Moreover, we can show that $\mathcal{P}^{exec}$ is equivalent to $\mathcal{P}$. Combining this result with the one of the previous section, we can conclude with the following theorem:

Theorem 18 *Let Δ be a set of full dependencies, $\mathcal{V}$ a set of conjunctive source descriptions with binding adornments, and let $\mathcal{Q}$ be a query over the relations in the mediated schema. Then $\bar{\mathcal{Q}} \cup chase(\Sigma) \cup \mathcal{T} \cup domain(\mathcal{V}, \mathcal{Q}) \cup (\mathcal{V}^{-1})^{exec}$ is maximally-contained in $\mathcal{Q}$ relative to Δ.* □

PROOF. The following two observations are used in the proof:

(C1) If $\mathcal{P}$ is an executable conjunctive plan for $\mathcal{Q}$, and X is a variable in $\mathcal{P}$, then any value that X can take during the execution of $\mathcal{P}$ will be in the extension of the predicate *dom*.

(C2) If $\mathcal{P}$ is an executable conjunctive plan, and $\mathcal{P}'$ is a reordering of the subgoals of $\mathcal{P}$ such that $\mathcal{P}'$ is also executable, then $\mathcal{P}$ and $\mathcal{P}'$ will produce the same set of answers.

The first claim is proved by induction on the place (i.e., subgoal number) in which X appears for the first time in $\mathcal{Q}$. The second claim is proved by showing that whenever a variable is bound for the first time (in either $\mathcal{Q}$ or $\mathcal{P}'$), it will be bound to a superset of the values it will have in the answer to the plan.

We denote the plan $\bar{\mathcal{Q}} \cup chase(\Sigma) \cup \mathcal{T} \cup domain(\mathcal{V}, \mathcal{Q}) \cup (\mathcal{V}^{-1})^{exec}$ by $\mathcal{P}_d$. The proof of the theorem proceeds as follows. From the previous theorems, we know that if we ignore the binding pattern limitations and the appearances of the predicate *dom* in $\mathcal{P}_d$, then for every conjunctive plan $\mathcal{P}$ that is contained in the query $\mathcal{Q}$, there exists a conjunctive plan $\mathcal{P}'$, that is one of the conjunctive queries encoded by $\mathcal{P}_d$, such that $\mathcal{P}$ is contained in $\mathcal{P}'$. Hence, there is a containment mapping ψ (that ignores the *dom* atoms in $\mathcal{P}'$) from the variables of $\mathcal{P}'$ to the variables of $\mathcal{P}$. To complete the proof, it suffices to show that during the execution of $\mathcal{P}'$ every subgoal g of $\mathcal{P}'$ will have at least the bindings that the subgoal $\psi(g)$ will have during the execution of $\mathcal{P}$, and that are in the final result of $\mathcal{P}$ (i.e., that are in the projection of the result of $\mathcal{P}$ on the variables in g). If we consider the subgoal g, there are two options. In the first, the binding of all the arguments of g come only from subgoals of the *dom* predicate. In this case, observation C1 entails that we have all the necessary bindings. The second option is that binding for some of the arguments of g come from previous subgoals in $\mathcal{P}'$. But in this case observation C2 entails that the ordering of the subgoals of $\mathcal{P}'$ does not change the result of $\mathcal{P}'$. □

Finally, we note that the query plan can be constructed in time polynomial in the size of $\mathcal{Q}$, $\mathcal{V}$ and Δ.

7 Eliminating function symbols

Although in Section 3 we demonstrated an efficient procedure to answer a datalog query as well as possible given only source relations, it is desirable to transform the constructed logic query to a datalog query that represents this answer. This means that we are looking for a datalog query that is equivalent to $(\mathcal{Q}^-, \mathcal{V}^{-1})\downarrow$. The key observation underlying the construction of such a datalog query is that there are only finitely many function symbols in $(\mathcal{Q}^-, \mathcal{V}^{-1})$. Because nested function expressions can never be generated using bottom-up evaluation, it is possible, with a little bit of bureaucracy, to keep track of function terms produced by $(\mathcal{Q}^-, \mathcal{V}^{-1})$ without actually generating tuples containing function terms.

The transformation proceeds in a bottom-up fashion. Function terms like $f(X_1, \ldots, X_k)$ in the IDB predicates of $\mathcal{V}^{-1}$ are eliminated by replacing them

by the list of variables $X_1, \ldots, X_k$ that occur in them. The IDB predicate names need to be annotated to indicate that $X_1, \ldots, X_k$ belonged to the function term $f(X_1, \ldots, X_k)$. For instance, in Example 6 the rule

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

is replaced by the rule

$$edge^{\langle \star, f(\star,\star)\rangle}(X, X, Y) :- s(X, Y)$$

The annotation $\langle \star, f(\star,\star)\rangle$ represents the fact that the first argument in $edge^{\langle \star, f(\star,\star)\rangle}$ is identical to the first argument in $edge$, and that the second and third argument in $edge^{\langle \star, f(\star,\star)\rangle}$ combine to a function term with the function symbol f as the second argument of $edge$. If bottom-up evaluation of $(\mathcal{Q}^-, \mathcal{V}^{-1})$ can yield a function term for an argument of an IDB predicate in $\mathcal{Q}^-$, then a new rule is added with correspondingly expanded and annotated predicates. The following definition states this construction formally. $\bar{X}$ is a shorthand for a list of variables or constants, and $\langle\bar{\beta}\rangle$ is a shorthand for an adornment. $\bar{X}[i]$ and $\bar{\beta}[i]$ stand for the ith position in $\bar{X}$ and $\bar{\beta}$ respectively.

Definition 19 (predicate splitting) *Let $\mathcal{P}$ be a query plan with function symbols. We are going to define a query plan $\mathcal{P}^{split}$ that encodes exactly the derivations in $\mathcal{P}$, but doesn't contain function symbols. The transformation from $\mathcal{P}$ to $\mathcal{P}^{split}$ is called* predicate splitting*, because an IDB predicate in $\mathcal{P}$ might be represented by several IDB predicates in $\mathcal{P}^{split}$.*

- *If*

  $p(\alpha_1, \ldots, \alpha_n) :- s(\bar{X})$

  is an inverse rule in $\mathcal{P}$, then the query plan $\mathcal{P}^{split}$ contains the rule

  $p^{\langle\beta_1,\ldots,\beta_n\rangle}(Y_1, \ldots, Y_{\gamma_n}) :- s(\bar{X})$

  with $\gamma_0 = 0$ and for $i = 1, \ldots, n$,

  $$|\alpha_i| = \begin{cases} 1 & : \text{ if } \alpha_i \text{ is a variable or a constant} \\ arity\ of\ f & : \text{ if } \alpha_i \text{ is a function term with function symbol } f, \end{cases}$$

  $\gamma_i = \gamma_{i-1} + |\alpha_i|$,

  $$\beta_i = \begin{cases} \star & : \text{ if } \alpha_i \text{ is a variable or a constant} \\ f(\underbrace{\star, \ldots, \star}_{|\alpha_i|}) & : \text{ if } \alpha_i \text{ is a function term with function symbol } f, \end{cases}$$

  and for $k_i = 1, \ldots, |\alpha_i|$:

$$Y_{\gamma_{i-1}+k_i} = \begin{cases} \alpha_i & : \text{ if } \alpha_i \text{ is a variable or a constant} \\ X & : \text{ if } \alpha_i \text{ is a function term with } X \text{ as its } k\text{th argument.} \end{cases}$$

- *If*

  $p(\bar{X}) :- p_1(\bar{X}_1),\ \ldots,\ p_m(\bar{X}_m)$

  is a rule in $\mathcal{P}$ *which is not an inverse rule, and*

  (i) the query plan $\mathcal{P}^{split}$ *contains rules that have* $p_1^{\langle\bar{\beta}_1\rangle}, \ldots, p_m^{\langle\bar{\beta}_m\rangle}$ *as heads,*

  (ii) if for some i, j, i', j', $\bar{X}_j[i]$ *is identical to* $\bar{X}_{j'}[i']$, *then* $\bar{\beta}_j[i] = \bar{\beta}_{j'}[i']$, *and*

  (iii) if for some i, j, $\bar{X}_j[i]$ *is a constant, then* $\bar{\beta}_j[i] = \star$,

  then the query plan $\mathcal{P}^{split}$ *contains the rule*

  $p^{\langle\bar{\beta}\rangle}(\bar{Y}) :- p_1^{\langle\bar{\beta}_1\rangle}(\bar{Y}_1),\ \ldots,\ p_m^{\langle\bar{\beta}_m\rangle}(\bar{Y}_m)$

  such that for all i, $\bar{\beta}[i] = \bar{\beta}_j[k]$ *for some* j, k *with* $\bar{X}[i] = \bar{X}_j[k]$, *and if a variable* X *that occurs at* $\bar{X}_j[k]$ *for some* j, k *occurs anywhere else, then the variables and constants that represent* X *in* $\bar{Y}_j$ *are the same as the variables and constants that represent* X *in the other places.*

The following example shows this transformation.

Example 20 *The logic query from example 6 is transformed to the following datalog query. The lines indicate the stages in the generation of the datalog rules.*

$$\begin{array}{ll}
edge^{\langle\star,f(\star,\star)\rangle}(X,X,Y) & :- s(X,Y) \\
edge^{\langle f(\star,\star),\star\rangle}(X,Y,Y) & :- s(X,Y) \\
\hline
q^{\langle\star,f(\star,\star)\rangle}(X,Y_1,Y_2) & :- edge^{\langle\star,f(\star,\star)\rangle}(X,Y_1,Y_2) \\
q^{\langle f(\star,\star),\star\rangle}(X_1,X_2,Y) & :- edge^{\langle f(\star,\star),\star\rangle}(X_1,X_2,Y) \\
\hline
q^{\langle\star,\star\rangle}(X,Y) & :- edge^{\langle\star,f(\star,\star)\rangle}(X,Z_1,Z_2),\ q^{\langle f(\star,\star),\star\rangle}(Z_1,Z_2,Y) \\
q^{\langle f(\star,\star),f(\star,\star)\rangle}(X_1,X_2,Y_1,Y_2) & :- edge^{\langle f(\star,\star),\star\rangle}(X_1,X_2,Z),\ q^{\langle\star,f(\star,\star)\rangle}(Z,Y_1,Y_2) \\
\hline
q^{\langle f(\star,\star),\star\rangle}(X_1,X_2,Y) & :- edge^{\langle f(\star,\star),\star\rangle}(X_1,X_2,Z),\ q(Z,Y) \\
q^{\langle\star,f(\star,\star)\rangle}(X,Y_1,Y_2) & :- edge^{\langle\star,f(\star,\star)\rangle}(X,Z_1,Z_2), \\
 & \quad q^{\langle f(\star,\star),f(\star,\star)\rangle}(Z_1,Z_2,Y_1,Y_2)
\end{array}$$

The generated datalog query shows explicitly in which arguments the original logic query was able to produce function terms. To the resulting datalog

program we can apply standard optimizations. For example, if a predicate p cannot contribute answers to the query (e.g., because in the predicate graph of the program there is no path from p to the query predicate), then the rules defining p can be deleted.

Example 21 *In the dependency graph for the datalog query in example 20, there are no paths from predicates* $q^{\langle \star, f(\star,\star)\rangle}$ *and* $q^{\langle f(\star,\star), f(\star,\star)\rangle}$ *to* q*. Therefore, these two predicates are irrelevant. The three rules defining the irrelevant predicates can be dropped. The following is the resulting datalog query:*

$$
\begin{aligned}
edge^{\langle \star, f(\star,\star)\rangle}(X, X, Y) &:- s(X, Y) \\
edge^{\langle f(\star,\star), \star\rangle}(X, Y, Y) &:- s(X, Y) \\
q^{\langle f(\star,\star), \star\rangle}(X_1, X_2, Y) &:- edge^{\langle f(\star,\star), \star\rangle}(X_1, X_2, Y) \\
q^{\langle f(\star,\star), \star\rangle}(X_1, X_2, Y) &:- edge^{\langle f(\star,\star), \star\rangle}(X_1, X_2, Z),\ q(Z, Y) \\
q^{\langle \star,\star\rangle}(X, Y) &:- edge^{\langle \star, f(\star,\star)\rangle}(X, Z_1, Z_2),\ q^{\langle f(\star,\star), \star\rangle}(Z_1, Z_2, Y)
\end{aligned}
$$

Because we keep track of function symbols in $(\mathcal{Q}^-, \mathcal{V}^{-1})^{split}$, we know that the resulting instance of the query predicate q with all "$\star$" adornment is exactly the subset of the result of $(\mathcal{Q}^-, \mathcal{V}^{-1})$ that does not contain function symbols. The following is therefore an immediate corollary of theorem 8.

Corollary 22 *For every datalog query* $\mathcal{Q}$ *and every set of conjunctive source descriptions* $\mathcal{V}$ *over the EDB predicates of* $\mathcal{Q}$*, the query plan* $(\mathcal{Q}^-, \mathcal{V}^{-1})^{split}$ *is maximally-contained in* $\mathcal{Q}$*. Moreover, if there exists a query plan that is equivalent to* $\mathcal{Q}$*, then* $(\mathcal{Q}^-, \mathcal{V}^{-1})^{split}$ *is equivalent to* $\mathcal{Q}$*.*

8 Conclusions

We introduced a novel approach to creating information gathering plans, that allows for recursive plans. We have shown that recursive plans enable us to solve three open problems. We described algorithms for obtaining a maximally-contained query plan in the case of recursive user queries, in the presence of dependencies and in the presence of limitations on binding patterns in the mediated schema. Our results are also of practical importance because dependencies and limitations on binding patterns occur very frequently in information sources in practice (e.g., the WWW).

Recursive information gathering plans have another important methodological advantage. Query plans can be constructed by *describing* a set of inferences that the mediator needs to make in order to obtain data from its sources. As

a consequence, it is simpler to construct these plans, and we believe that it is easier to extend our methods to other contexts.

Acknowledgements

We thank Harish Devarajan and Dan Weld for discussions and comments on earlier versions of this article.

References

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '98*, June 1998.

[2] S. Adalı, K. S. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 137–148, June 1996.

[3] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM Journal on Computing*, 8(3):218–246, May 1979.

[4] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query reformulation for dynamic information integration. *International Journal on Intelligent and Cooperative Information Systems*, 6(2/3):99–130, June 1996.

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

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

[7] S. Chaudhuri and M. Y. Vardi. One the equivalence of recursive and nonrecursive datalog programs. *Journal of Computer and System Sciences*, 54(1):61 – 78, Feb. 1997.

[8] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the 100th Anniversary Meeting*, pages 7–18, Tokyo, Japan, Oct. 1994. Information Processing Society of Japan.

[9] O. M. Duschka. Query optimization using local completeness. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, pages 249–255, Providence, RI, July 1997.

[10] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '97*, pages 109 – 116, Tucson, AZ, May 1997.

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

[12] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI*, Nagoya, Japan, Aug. 1997.

[13] O. Etzioni, K. Golden, and D. Weld. Tractable closed world reasoning with updates. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 178–189, San Francisco, CA, June 1994.

[14] O. Etzioni, K. Golden, and D. Weld. Sound and efficient closed-world reasoning for planning. *Journal of Artificial Intelligence*, 89(1–2):113–148, Jan. 1997.

[15] O. Etzioni and D. S. Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76, 1994.

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

[17] M. Friedman and D. S. Weld. Efficiently executing information-gathering plans. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI*, Nagoya, Japan, Aug. 1997.

[18] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

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

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

[21] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 402–412, Bombay, India, 1996.

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

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

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

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

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

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

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

[29] A. Tomasic, L. Raschid, and P. Valduriez. A data model and query processing techniques for scaling access to distributed heterogeneous databases in Disco. *IEEE Transactions on Computers, special issue on Distributed Computing Systems*, 1997.

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

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

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