Recursive Plans for Information Gathering

Oliver M. Duschka
Department of Computer Science
Stanford University
Stanford, California 94305
U.S.A.

Alon Y. Levy
AT&T Laboratories
600 Mountain Avenue
Murray Hill, New Jersey 07974
U.S.A.

Abstract
Generating query-answering plans for information gathering agents requires to translate a user query, formulated in terms of a set of virtual relations, to a query that uses relations that are actually stored in information sources. Previous solutions to the translation problem produced sets of conjunctive plans, and were therefore limited in their ability to handle information sources with binding-pattern limitations, and to exploit functional dependencies in the domain model. 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 information gathering plans, which enables us to settle two open problems. First, we describe an algorithm for finding a query plan that produces the maximal set of answers that can be obtained from the sources in the presence of functional dependencies. Second, we describe an analogous algorithm in the presence of binding-pattern restrictions in the sources, which was not possible without recursive plans.

1 Introduction
The problem of information integration (a.k.a. information gathering agents) has recently received considerable attention due to the growing number of structured information sources available online. Information integration systems (e.g., the Internet Softbot [Etzioni and Weld, 1994], SIMS [Arens et al., 1996], TSIMMIS [Chawathe et al., 1994], the Information Manifold [Levy et al., 1996], Occam [Kwok and Weld, 1996], Information Master [Duschka and Genesereth, 1997b]) provide a uniform query interface to the multiple information 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.

Information integration systems are based on the following general architecture. The user interacts with a uniform interface in the form of a set of virtual relation names that are used in formulating queries. 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 virtual relations and the source relations. The most common method to specify these mappings (employed in [Levy et al., 1996; Kwok and Weld, 1996; Duschka and Genesereth, 1997b]) is to describe each source relation as the result of a conjunctive query (i.e., a single Horn rule) over the virtual relations. For example, an information source containing papers authored by AI researchers would be described as follows:

\[ db_1(P, A) : = \text{paper}(P), \text{author}(P, A), \text{ai}(A). \]

The relations paper, author and ai are virtual relations that can be used in formulating queries, and db_1 is a source relation.

Given a query from the user, formulated in terms of the virtual relations, the system must translate it to a query that mentions only the source relations, because only those relations are actually available. That is, the system needs to find a query expression, (or, a set of conjunctive queries), that mentions only the source relations, and is equivalent to the original query. The new query is called a query plan (or a query rewriting), and the translation problem is known as the problem of rewriting queries using views. The views (a term used in database systems to refer to predefined queries) are the relations in the sources. The rewriting problem has also been investigated in the database literature because of its importance for query optimization and data warehousing [Yang and Larson, 1987; Chaudhuri et al., 1995; Levy et al., 1995; Rajaraman et al., 1995; Qian, 1996; Duschka and Genesereth, 1997a].

Previous results on the rewriting problem are of lim-
ited practical use for several reasons. First, they have concentrated on showing a bound on the size of the resulting query plan [Levy et al., 1995; Rajaraman et al., 1995]. These results establish the complexity of the rewriting problem, but yield only non-deterministic algorithms for its solution. Second, they considered only the problem of finding an equivalent rewriting of the query using the source relations. In practice, the collection of available sources may not contain all the information needed to answer a query, and therefore, we need to resort to maximally-contained rewritings. A maximally-contained rewriting provides all the answers that are possible to obtain from the sources, but the expression describing the rewriting may not be equivalent to the original query. For example, if we only have the db1 source available, and our query asks for all papers by Computer Science researchers, then the following is a maximally-contained rewriting:

\[
q(P,Y,A) : \leftarrow \text{db1}(P,Y,A).
\]

The third problem is that many sources encountered in practice have limitations on the binding patterns they support, or may satisfy certain functional dependencies. As an example of binding-pattern restrictions, 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. In the case of equivalent rewritings, Rajaraman et al. [Rajaraman et al., 1995] describe a bound on the size of the query plans that need to be considered. However, Kwok and Weld [Kwok and Weld, 1996] show that if we restrict our plans to be sets of conjunctive queries, then there may not be a finite maximally-contained rewriting. As an example of a functional dependency, the year of a conference functionally determines its location. The presence of functional dependencies further complicates the rewriting problem because it allows rewritings that are not valid otherwise. By ignoring the functional dependencies, we may miss answers to the query.

In this paper we introduce the new class of recursive query plans for information gathering. Instead of plans being only sets of conjunctive queries, they can now be recursive sets of function-free Horn rules. Using recursive plans, we are able to settle two open problems. First, we describe an algorithm for finding the maximally-contained rewriting in the presence of functional dependencies. Second, we describe an algorithm for finding the maximally-contained rewriting in the presence of binding-pattern restrictions, which was not possible without recursive plans.

Another significant advantage of our method is that it is generative, rather than descriptive. Our algorithms generate the rewriting in time that is polynomial in the size of the query. In contrast, previous methods [Levy et al., 1995; Rajaraman et al., 1995] describe the space of possible candidate rewritings, and propose heuristics for searching this space [Kwok and Weld, 1996; Levy et al., 1996]. 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.

2 Preliminaries

Relations and queries: Our representation of the domain and of the information sources includes a set 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, we denote by l(A) the value of the attribute A in t.

A function-free Horn rule is an expression of the form

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

where \(p_i\) and \(p_1, \ldots , p_n\) are relation names, and \(\bar{x}_1, \ldots , \bar{x}_n\) are tuples of variables and constants such that any variable appearing in \(\bar{X}\) appears also in \(\bar{x}_1 \cup \ldots \cup \bar{x}_n\). The head of the rule is \(p(\bar{X})\), and its body is \(p_1(\bar{x}_1), \ldots , p_n(\bar{x}_n)\). The base relations of a set of Horn rules are the relations that appear only in the bodies of the rules and not in the heads (and therefore their extension includes only explicitly stored tuples). Given a set of rules, we can define a dependency graph, whose nodes are the relations appearing in the rules. There is an arc from the node of relation \(p_i\) to the node of predicate \(p\) if \(p_i\) appears in the body of a rule whose head relation is \(p\). The rules are said to be recursive if there is a cycle in the dependency graph.

A query is a set of function-free Horn rules. A conjunctive query is a single non-recursive Horn rule. One of the relations in the query is distinguished as the query relation, and its extension is the answer to the query.

Query containment: In our discussion we will need to compare between different queries. We use the condition of containment to compare queries. Let us denote by \(q(D)\) the result of evaluating query \(q\) on database \(D\). Given two queries \(q_1\) and \(q_2\), we say that \(q_1\) is contained in \(q_2\) if for every database \(D\), \(q_1(D) \subseteq q_2(D)\). The fundamental results on containment of conjunctive queries

\[1\]The algorithm in [Levy et al., 1996] 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 [Kwok and Weld, 1996] 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.
and function-free Horn rules are described in [Chandra and Merlin, 1977; Sagiv and Yannakakis, 1980; Shmueli, 1993].

**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 \( \Sigma \), we refine our notion of containment to *relative containment.* Query \( q_1 \) is contained in query \( q_2 \) relative to \( \Sigma \), denoted \( q_1 \subseteq_{\Sigma} q_2 \), if for each database \( D \) satisfying the functional dependencies in \( \Sigma \), \( q_1(D) \subseteq q_2(D) \).

In order to decide containment of conjunctive queries in the presence of functional dependencies, Aho et al. [Aho et al., 1979] show that it suffices to precede the containment algorithm by applying the chase algorithm to the contained query. The chase algorithm applies the following transformation to the body of a conjunctive query \( q \) until no changes can be made. If the functional dependency \( \bar{A} \rightarrow B \) holds for a relation \( p \), and a conjunctive query \( q \) 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 \( q \) by the value or variable for \( B \) in \( g_2 \).

**Modeling information sources and query plans:** The domain model of an information agent is a set of *virtual relations.* The relations are virtual because they are only meant to provide the user a uniform interface to a multitude of information sources, and the agent does not actually store the extensions of these relations. In our discussion, we assume that a user query is a conjunctive query over the virtual relations.\(^2\)

The agent models the contents of the external information sources by a set of *source relations,* that are disjoint from the virtual relations. To answer user queries, the agent must also have a mapping between the virtual and source relations. The mappings, called *source descriptions,* are specified by a set of conjunctive queries, whose bodies contain only virtual relations and their heads are source relations. The meaning of such a mapping is that all the tuples that are found in the information source satisfy the query over the virtual relations.\(^3\) Given a query \( q \) from the user, the agent needs to formulate a query plan, which is a query that bottoms out in the source relations and produces answers to \( q \). A query plan is a set of Horn rules whose base predicates include only the source relations.

**Example 2.1:** Consider a domain model where *parent, male* and *female* are virtual relations. The mappings below say that the source relations \( v_1 \) and \( v_2 \) store the father and mother relation, respectively.

\[
\begin{align*}
v_1(X,Y) & : = \text{parent}(X,Y), \text{male}(X) \\
v_2(X,Y) & : = \text{parent}(X,Y), \text{female}(X)
\end{align*}
\]

The following query plan determines all grandparents of \( \text{ann} \) from the available sources:

\[
\begin{align*}
\text{answer}(X) & : = \text{parent}(X,Z), \text{parent}(Z,\text{ann}) \\
\text{parent}(X,Y) & : = v_1(X,Y) \\
\text{parent}(X,Y) & : = v_2(X,Y)
\end{align*}
\]

The *expansion* of a query plan \( \mathcal{P} \), denoted \( \mathcal{P}^{\exp} \), is obtained from \( \mathcal{P} \) by replacing all source-relation literals by their definitions. Existentially quantified variables in a source description are replaced by fresh variables in the expansion.

A query plan \( \mathcal{P} \) is *maximally-contained* in a query \( q \), relative to a set of functional dependencies \( \Sigma \), if \( \mathcal{P}^{\exp} \subseteq_{\Sigma} q \), and for every query plan \( \mathcal{P}' \), if \( \mathcal{P}'^{\exp} \subseteq_{\Sigma} q \) then \( \mathcal{P}'^{\exp} \subseteq_{\Sigma} \mathcal{P}^{\exp} \).

### 3 Functional Dependencies

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 virtual relations

\[
\begin{align*}
\text{conference}(\text{Paper,Conference}) \\
\text{year}(\text{Paper, Year}) \\
\text{location}(\text{Conference, Year, Location})
\end{align*}
\]

describing the conference at which a paper was presented, the publication year of a paper, and the location a conference was held 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:

\[
\begin{align*}
\text{conference} & : \text{Paper} \rightarrow \text{Conference} \\
\text{year} & : \text{Paper} \rightarrow \text{Year} \\
\text{location} & : \text{Conference, Year} \rightarrow \text{Location}
\end{align*}
\]

We have the following information sources:

\[
\begin{align*}
v_1(P,C,Y) & : = \text{conference}(P,C), \text{year}(P,Y) \\
v_2(P,L) & : = \text{conference}(P,C), \text{year}(P,Y), \text{location}(C,Y,L)
\end{align*}
\]

\( v_1 \) tells us in which conference and year a paper was presented, and \( v_2 \) stores the location of the presentation of a paper directly with the paper. Assume a user wants to know where IJCAI '91 was held:

\[
\begin{align*}
\text{answer}(X) & : = v_1(X,Y) \\
\text{location}(X,Y,Z) & : = v_2(X,Y,Z)
\end{align*}
\]

\( v_1 \) and \( v_2 \) are the *source relations.* The *query plan* \( \mathcal{P} \) is as follows:

\[
\begin{align*}
\text{answer}(X) & : = v_1(X,Y) \\
\text{location}(X,Y,Z) & : = v_2(X,Y,Z)
\end{align*}
\]

\( \mathcal{P} \) is *maximally-contained* relative to the above dependencies.

\(^2\)Our results apply also to the case in which user queries are recursive [Dischka and Genesereth, 1997a].

\(^3\)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 [Ettzioni et al., 1994; Levy, 1996]. For our discussion this distinction does not matter.
\( q(L) \leftarrow location(ijcai, 1991, L) \)

The following plan would answer the query:

\( answer(L) \leftarrow v_1(P, ijcai, 1991), v_2(P, L) \)

The query plan finds some paper presented at IJCAI '91 using \( v_1 \), and then finds the location of the conference this paper was presented at using \( v_2 \). 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 sources. It is also important to note that view \( v_1 \) is needed in the query plan even though the predicates in \( v_1 \), \textit{conference} and \textit{year}, don't appear in the query \( q \) at all. Without functional dependencies, only views that contain predicates appearing in the user query need to be considered [Levy \textit{et al.}, 1995].

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. The key to the construction is a set of inverse rules, whose purpose is to recover tuples of the virtual relations from the source relations. In the following definition we use a set of function symbols; for every source relation \( v \) with variables \( X_1, \ldots, X_n \) in the body but not in the head of the source description, we have a function symbol \( f_{v,i} \). These function symbols can later be removed from the query plan\(^4\) [Duschka and Genesereth, 1997a].

**Definition 3.1:** (inverse rules) Let \( v \) be a source description \( v(X) := p_1(X_1), \ldots, p_n(X_n) \). Then for \( j = 1, \ldots, n \),

\[ p_j(X_j) := v(X) \]

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

We denote the set of inverse rules of the views in \( V \) by \( V^{-1} \).

Continuing with our example, the inverse rules for \( v_1 \) and \( v_2 \) are:

\[ \begin{align*}
    r_1: \textit{conference}(P, C) & \leftarrow v_1(P, C, Y) \\
    r_2: \textit{year}(P, Y) & \leftarrow v_1(P, C, Y) \\
    r_3: \textit{conference}(P, f_1(P, L)) & \leftarrow v_2(P, L) \\
    r_4: \textit{year}(P, f_2(P, L)) & \leftarrow v_2(P, L) \\
    r_5: \textit{location}(f_1(P, L), f_2(P, L), L) & \leftarrow v_2(P, L)
\end{align*} \]

For example, rule \( r_5 \) extracts from \( v_2 \) that some conference in some year was held in location \( L \). Suppose that \( v_1 \) stores the information that the paper "Fuzzy Dynamic Systems" was presented at IJCAI '91, and \( v_2 \) stores the information that "Fuzzy Dynamic Systems" was presented in Sydney. The inverse rules derive the following facts:

\[ \begin{align*}
    \textit{conference} & (\textit{fuzzy}, \textit{ijcai}) \\
    \langle \textit{fuzzy}, f_1(\textit{fuzzy}, \textit{sydney}) \rangle & (\text{with } r_1) \\
    \textit{year} & (\textit{fuzzy}, 1991) \\
    \langle \textit{fuzzy}, f_2(\textit{fuzzy}, \textit{sydney}) \rangle & (\text{with } r_2) \\
    \textit{location} & (f_1(\textit{fuzzy}, \textit{sydney}), f_2(\textit{fuzzy}, \textit{sydney}), \textit{sydney}) (r_5)
\end{align*} \]

The inverse rules don't take into account the presence of the functional dependencies. For example, because of the functional dependency in relation \textit{conference}, \textit{Paper} \( \rightarrow \) \textit{Conference}, it is possible to conclude that the function term \( f_1(\textit{fuzzy}, \textit{sydney}) \) must actually be the same as the constant \textit{ijcai}. 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 conclusion of \( \textit{e} \) includes the extension of \( = \) (i.e., for every \( X \), \( e(X, X) \)), and the tuples that can be derived by the following chase rules \( e(\bar{A}, \bar{B}) \) is a shorthand for \( e(A_1, A'_1), \ldots, e(A_n, A'_n) \).\(^5\)

**Definition 3.2:** (chase rules) Let \( \bar{A} \rightarrow B \) be a functional dependency satisfied by a virtual relation \( p \). Let \( C \) be the attributes of \( p \) that are not in \( \bar{A}, B \). The chase rule corresponding to \( \bar{A} \rightarrow B \), denoted \( \textit{chase}(\bar{A} \rightarrow B) \), is the following rule:

\[ e(B, B') := p(\bar{A}, B, C), p(\bar{A}', B', C'), e(\bar{A}, \bar{A}') \]

We denote by \( \textit{chase}(\Sigma) \) the set of chase rules corresponding to the functional dependencies in \( \Sigma \).

In our example, the chase rules are:

\[ \begin{align*}
    e(C, C') & := \textit{conference}(P, C), \textit{conference}(P', C'), e(P, P') \\
    e(Y, Y') & := \textit{year}(P, Y), \textit{year}(P', Y'), e(P, P') \\
    e(L, L') & := \textit{location}(C, Y, L), \textit{location}(C', Y', L'), e(C, C'), e(Y, Y')
\end{align*} \]

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

\( \)\(^5\)We only require 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.
The extension of $\epsilon$ is reflexive by construction, and is symmetric because of the symmetry in the chase rules. To guarantee that $\epsilon$ is an equivalence relation, it is still needed to enforce transitivity of $\epsilon$. The following rule, denoted by $T$, is sufficient for guaranteeing transitivity of relation $\epsilon$:

$$\epsilon(X, Y) \rightarrow \epsilon(X, Z), \epsilon(Z, Y)$$

The final step in the construction is to rewrite query $q$ in a way that it can use the equivalences derived in relation $\epsilon$.

We define the query $\bar{q}$ by modifying $q$ iteratively as follows. If $\epsilon$ is a constant in one of the subgoals of $q$, we replace it by a new variable $\bar{Z}$, and add the subgoal $\epsilon(\bar{Z}, \epsilon)$. If $X$ is a variable in the head of $q$, we replace it by a new variable $\bar{Y}$, and add the subgoal $\epsilon(\bar{X}, X)$. If a variable $Y$ that is not in the head of $q$ appears in two subgoals of $q$, we replace one of its occurrences by $\bar{Y}$, and add the subgoal $\epsilon(Y, \bar{Y})$. We continue until we cannot apply this rule anymore. Our example query would be rewritten to

$$\bar{q}(L) := \text{location}(C, V, L), \epsilon(C, \text{ijcai}), \epsilon(Y, 1991), \epsilon(L, L)$$

Note that evaluating query $\bar{q}$ on the reconstructed virtual relations and the derived equivalence relation $\epsilon$ yields the desired result: IJCAI '91 was held in Sydney.

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

**Theorem 3.1:** Let $\Sigma$ be a set of functional dependencies, $\mathcal{V}$ a set of conjunctive source descriptions, and let $q$ be a conjunctive query over the virtual relations. Let $\mathcal{R}$ denote the set of rules $\mathcal{V}^{-1} \cup \text{chase}(\Sigma) \cup T$. Then, $\mathcal{Q} \cup \mathcal{R}$ is maximally-contained in $q$ relative to $\Sigma$. Furthermore, $\mathcal{Q} \cup \mathcal{R}$ can be constructed in time polynomial in the size of $q$, $\mathcal{V}$, and $\Sigma$. $\square$

**Proof:** The key to the proof is to show that for every conjunctive query plan $\mathcal{P} \subseteq q$, $\mathcal{P}^{\text{exp}} \subseteq (\mathcal{Q} \cup \mathcal{R})^{\text{exp}}$. 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 $q$ is a query, $\mathcal{P}$ is a conjunctive query plan, and $e_1, \ldots, e_k$ is a sequence of queries with $e_1 = \mathcal{P}^{\text{exp}}$, $e_k \subseteq q$, and $e_{k+1}$ results from $e_k$ by applying a chase step, then $\mathcal{P}^{\text{exp}} \subseteq (\mathcal{Q} \cup \mathcal{R})^{\text{exp}}$. This would prove that $\mathcal{Q} \cup \mathcal{R}$ is maximally-contained in $q$ relative to $\Sigma$.

For $k = 1$, $\mathcal{P}^{\text{exp}}$ is contained in $q$. As shown in [Duschka and Genesereth, 1997a], this implies that $\mathcal{P}^{\text{exp}}$ is contained in $(q \cup \mathcal{V}^{-1})^{\text{exp}}$. It follows that $\mathcal{P}^{\text{exp}}$ is contained in $(\mathcal{Q} \cup \mathcal{R})^{\text{exp}}$ relative to $\Sigma$.

For the induction step, let $k > 1$ and assume $e_{k-1} \subseteq q$. Let $\mathcal{A} \rightarrow \mathcal{B}$ be the functional dependency that holds for relation $\mathcal{P}$ 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 $\mathcal{A}$, 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 [Chandra and Merlin, 1977] that shows that $q$ contains $e_k$. Replace every value/variable $X_i$ in an argument position in $q$ that is mapped by $h$ to an argument position in $e_k$ that used to be variable $X_i$ in $e_{k-1}$ by a new variable $X'_i$. For each of the new variables $X'_i$, add two subgoals of $p$ to $q$ with the identical new variables for the corresponding attributes $\mathcal{A}$, $X_i$, and $X'_i$ for the remaining attributes. We can now find a containment mapping from query $q'$ to query $e_{k-1}$. This shows that $e_{k-1}$ is contained in $q'$. Therefore, $\mathcal{P}^{\text{exp}} \subseteq q'$. By the induction hypothesis we have that $\mathcal{P}^{\text{exp}} \subseteq (\mathcal{Q} \cup \mathcal{R})^{\text{exp}}$. Using the chase rule $\text{chase}(\mathcal{A} \rightarrow \mathcal{B})$, the transitivity rule, and the reflexivity of relation $\epsilon$, we can show that $\mathcal{Q} \cup \mathcal{R} \subseteq (\mathcal{Q} \cup \mathcal{R})^{\text{exp}}$. It follows that $\mathcal{P}^{\text{exp}} \subseteq ((\mathcal{Q} \cup \mathcal{R})^{\text{exp}})$. Query $\bar{q}$ contains all subgoals in $q$, and at most as many additional subgoals of $\epsilon$ as the sum of all arities of the subgoals in $q$. 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 $\Sigma$. We can conclude that $\mathcal{Q} \cup \mathcal{R}$ can be constructed in time polynomial in the size of $q$, $\mathcal{V}$ and $\Sigma$. $\square$

### 4 Limitations on binding patterns

Some information sources cannot answer arbitrary atomic queries on the relation they contain. To model source capabilities, we attach to each source relation an adormion (see [Ullman, 1989], Chap. 12), specifying which binding patterns the source supports. An adormion of a source relation $v$ is a string of $b$'s and $f$'s of length $n$, where $n$ is the arity of $v$. The meaning of the adormion is that the source only supports queries in which the arguments with $b$ adormions are bound. The other arguments may be either bound or free. For example, the adormion $fb$ means that the first argument must be bound in queries on $v$. We define an executable Horn rule as follows.

---

6 For simplicity of exposition, we assume that each source relation has a single adormion.
Definition 4.1: (executable Horn rule) Let \( \mathcal{V} \) be a set of relations with binding adornment, and let \( r \) be the following Horn rule whose body relations are in \( \mathcal{V} \):

\[
q(X) := v_1(X_1), \ldots, v_n(X_n)
\]

The rule \( r \) is executable if the following holds for \( i = 1, \ldots, n \): let \( j \) be an argument position of \( v_i \) that has a b adornment, and let \( \alpha \) be the \( j \)th element in \( X_i \). Then, either \( \alpha \) is a constant, or \( \alpha \) appears in \( X_1 \cup \ldots \cup X_{i-1} \).

A query plan includes source relations and other relations, which we model as having the all-free adornment (i.e., \( f^0 \), 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.

When sources have limitations on binding patterns, it turns out that there may not be a finite maximally-contained plan, if we restrict ourselves to plans without recursion. The following example, adapted from [Kwok and Weld, 1996], illustrates the point.

Example 4.1: Consider the following sources:

\[
\begin{align*}
v_1^i(X) & := ijcapiPapers(X) \\
v_2^j(X, Y) & := cites(X, Y) \\
v_3^j(X) & := awardPaper(X).
\end{align*}
\]

The first source stores IJCAI 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:

\[
q(X) := \text{awardPaper}(X)
\]

For each \( n \), the following is an executable conjunctive query plan that is contained in \( q \):

\[
q_n(Z_n) := v_1(Z_0), v_2(Z_0, Z_1), \ldots, v_3(Z_{n-1}, Z_n),
\]

Furthermore, for each \( n \), \( q_n \) may produce answers that are not obtained by any other \( q_i \), for any \( i \). Intuitively, a paper will be in the answer to \( q_i \) if the number of links that need to be followed from an IJCAI 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 plan. On our example, the construction will yield the following query plan. The construction is based on inventing a new recursively-defined relation, \( \text{papers} \), whose extension will be the set of all papers that can be reached from the papers in \( v_1 \).

\[
\begin{align*}
\text{papers}(X) & := v_1^i(X) \\
\text{papers}(X) & := \text{papers}(Y), v_2^j(Y, X) \\
q(X) & := \text{papers}(X), v_3^j(X).
\end{align*}
\]

We now describe the construction for a given set of adorned source relations \( \mathcal{V} \) and a query \( q \). The recursive plan includes a unary relation \( \text{dom} \) 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 \( \text{dom} \) are the following.

Definition 4.2: (domain rules) Let \( v \in \mathcal{V} \) be a source relation of arity \( n \). Suppose the adornment of \( v \) 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:

\[
\begin{align*}
dom(X_i) & := \text{dom}(X_1), \ldots, \text{dom}(X_l), \\
v(X_1, \ldots, X_n).
\end{align*}
\]

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

We denote by \( \text{domain}(\mathcal{V}, q) \) the set of rules described above for defining the predicate \( \text{dom} \). Notice that all domain rules are executable, and that relation \( \text{dom} \) has adornment \( f \). Every query plan \( \mathcal{P} \) can be transformed to an executable query plan by inserting the literal \( \text{dom}(X) \) 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 \( \mathcal{P}^{\text{exec}} \), is executable. Moreover, we can show that \( \mathcal{P}^{\text{exec}} \) is equivalent to \( \mathcal{P} \). Combining this result with the one of the previous section, we can conclude with the following theorem:

Theorem 4.1: Let \( \Sigma \) be a set of functional dependencies, \( \mathcal{V} \) a set of conjunctive source descriptions with binding adornments, and let \( q \) be a conjunctive query over the virtual relations. Then \( \mathcal{P}^{\text{exec}}(\Sigma) \cup \text{domain}(\mathcal{V}, q) \cup (\Sigma^{-1})^{\text{exec}} \) is maximally-contained in \( q \) relative to \( \Sigma \).

Finally, we note that the query plan can be constructed in time polynomial in the size of \( q, \mathcal{V} \) and \( \Sigma \).

5 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 two open problems. We described algorithms for obtaining a maximally-contained query plan in the presence of functional dependencies and in the presence of limitations on binding patterns. Our results are also of practical importance because functional 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 information agent needs to make in order to obtain data from its sources. We are currently extending our algorithms to deal with order predicates (e.g., \( \leq, <, \neq \)) and with local completeness information about the sources [Etzioni et al., 1994; Duschka, 1997].
Acknowledgements
We thank Harish Devarajan and Dan Weld for discussions and comments on earlier versions of the paper.

References


