Answering Queries with Useful Bindings

Chen Li
Stanford University
and
Edward Chang
University of California, Santa Barbara

In information-integration systems, sources may have diverse and limited query capabilities. To obtain maximum information from these restrictive sources to answer a query, one can access sources that are not specified in the query (i.e., off-query sources). In this paper, we propose a query-planning framework to answer queries in the presence of limited access patterns. In the framework, a query and source descriptions are translated to a recursive datalog program. We then solve optimization problems in this framework, including how to decide whether accessing off-query sources is necessary, how to choose useful sources for a query, and how to test query containment. We develop algorithms to solve these problems, and thus construct an efficient program to answer a query.

Categories and Subject Descriptors: H.2.4 [**Information Systems**]: Systems—*Query Processing*; H.2.5 [**Information Systems**]: Heterogeneous Databases

General Terms: Algorithms, Performance

Additional Key Words and Phrases: information-integration systems, limited source capabilities, datalog programs, query containment

1. INTRODUCTION

The goal of information integration is to support seamless access to heterogeneous data sources. Many systems (e.g., TSIMMIS [Chawathe et al. 1994], the Informa-

This journal article combines and integrates work presented in The 16th International Conference on Data Engineering, San Diego, CA, March, 2000 [Li and Chang 2000], and The 8th International Conference on Database Theory, London, United Kingdom, January, 2001 [Li and Chang 2001]. In addition to the prior materials, this article contains theorem proofs that were not included in the original papers.
Name: Chen Li
Address: Computer Science Department, Stanford, CA 94305, chenli@cs.stanford.edu
Name: Edward Chang
Address: Electrical & Computer Engineering, University of California, Santa Barbara, CA 93106, echang@ece.ucsb.edu

tion Manifold [Levy et al. 1996], Garlic [Carey et al. 1995], Infomaster [Genesereth et al. 1997], Disco [Tomasic et al. 1998], Tukwila [Ives et al. 1999], and InfoSleuth [Bayardo Jr. et al. 1997]) have been proposed to reach this goal. To perform queries on sources, many studies [Abiteboul and Duschka 1998; Duschka 1998; Levy et al. 1995; Qian 1996; Rajaraman et al. 1995] construct answers to queries using views. These approaches are closely related to query-containment algorithms for conjunctive queries and for datalog programs [Ullman 1997].

In many heterogeneous environments, such as the World Wide Web, information sources may have diverse and limited query capabilities. Use some popular bookseller Web sites as an example. Amazon.com and Barnesandnoble.com do not allow users to freely download all records in their sources. Instead, they provide forms for users to fill out attributes such as book title, author name, publisher and ISBN, and return the records that satisfy the query conditions. In this paper, we show that we can use the sources not specified in a query (i.e., off-query sources) to obtain more information from restrictive sources, and compute answers to the query, as illustrated by the following example.

EXAMPLE 1. *Suppose we want to compare the average prices of the books sold by amazon.com and barnesandnoble.com. Since both sources require each source query to specify at least an ISBN, an author, or a title, we cannot retrieve all their book records. Suppose we can access prenhall.com to retrieve all authors who have published books through the publisher. We can use this author list to query amazon.com and barnesandnoble.com to get books and their prices and to compute the average prices.*

Note that the authors who publish books through Prentice Hall may also publish books through other publishers. We can use the author list of Prentice Hall as a seed list to retrieve book titles from the two bookstore sources, then use these titles to retrieve more authors, and so on so forth. We repeat the iteration until the author list remains stable, then average the prices of the books from the two sources.

Example 1 suggests that we can use the source prenhall.com to retrieve bindings for the author domain to answer the query, although this source is not mentioned in the query. In some cases, as shown by a more complex example in Section 2, we can access sources repeatedly to obtain bindings to compute more results for a query. In this paper we propose a query-planning framework for dealing with information-integration systems with source restrictions. In the framework, source descriptions and a query are translated into a datalog program [Ullman 1989], which can be evaluated on the source relations to answer the query.[1]

This framework exhibits two challenging problems. First, we need to decide when accessing off-query sources is necessary. Some of these accesses are essential, as they provide bindings that let us query sources, which we could not do otherwise. However, some accesses can be proven not to add anything to the query's answer. We show in what cases off-query accesses are necessary, and develop an algorithm for finding all the relevant sources for a query.

Second, we need to determine whether the maximal answer to a query is contained in that to another query. Since our framework often produces a recursive

[1]The idea is borrowed from [Duschka and Levy 1997]. In Section 1.1 we show the differences.

datalog program to answer a query optimally, and containment of datalog programs is undecidable [Shmueli 1993], our containment problem seems to be undecidable. (Our containment problem uses set semantics, not bag semantics.) We show that this containment problem is decidable since it can be reduced to containment of monadic programs, which is known to be decidable [Cosmadakis et al. 1988]. In addition, we introduce the question of *boundedness* for the programs in our framework. When one of the two programs in the containment test is bounded (i.e., it is equivalent to a finite union of conjunctive queries), the containment test can be performed efficiently [Chandra and Merlin 1977; Chaudhuri and Vardi 1992; Sagiv and Yannakakis 1980]. We develop a polynomial-time algorithm for testing query boundedness.

In this study we focus on a class of *connection queries*. A connection query is a natural join of distinct source views with the necessary selection and projection. (The details are described in Section 2.) Here we are taking the following universal-relation-like assumption [Ullman 1989]: different attributes that share the same name in different views have the same meaning. However, universal-relation study did not consider restrictions of retrieving information from relations. As we will see in Section 2.2, a connection query can be generated in various cases, where our techniques are applicable. The reason we study connection queries is that it is relatively easier (compared to arbitrary conjunctive queries) to trim useless source relations, and test connection boundedness. In Section 7 we discuss how to extend our results on connection queries to conjunctive queries.

The rest of the paper is organized as follows. Section 2 gives our motivating example and introduces the notation used throughout the paper. In Section 3, we propose a query-planning framework in integration systems with source restrictions. In the framework, source descriptions and a query are translated into a datalog program, which can be evaluated on the sources to compute the maximal answer to the query. In Sections 4 and 5, we consider each connection in a query individually, and solve the problem of trimming useless source relations for a connection. In Section 4, we discuss in what cases accessing off-connection views is necessary to answer a connection. In Section 5, we develop a polynomial-time algorithm for finding all the relevant sources for a connection. In Section 6, we show how to test whether the answer to one connection is contained in that to another connection. In such a case, the source accesses in the contained connection can be saved. Finally, we offer our conclusions and discussions in Section 7.

1.1 Related work

There are two approaches to information integration [Duschka 1998]:

(1) The source-centric approach: Both user queries and source views are in terms of some global views. For each query, the integration system needs to plan how to answer the query using source views. The Information Manifold and Infomaster follow this approach.

(2) The query-centric approach: User queries are in terms of views synthesized at a mediator [Wiederhold 1992] that are defined on source views. After view expansion [Li et al. 1998] at the mediator, the query is translated to a logical plan that is composed of the source views. TSIMMIS follows this approach,

and we follow this approach in this paper.

Ullman [1997] gave a good survey on the differences between these two approaches. Many studies have been done by taking the source-centric approach. For example, Qian [1996] discussed how to use query folding to rewrite queries using views without considering source restrictions. Duschka and Genesereth [1997] studied the problem of answering datalog queries using views, and the plan can be a datalog program. Rajaraman, Sagiv, and Ullman [1995] proposed algorithms for answering queries using views [Levy et al. 1995] with binding patterns. Duschka and Levy [1997] considered source restrictions by translating source binding patterns into rules in a datalog program, assuming that all attributes share one domain. We borrow the idea in that paper that uses domain predicates to construct datalog programs. We introduce one predicate for each domain, while [Duschka and Levy 1997] uses one predicate for all domains. In addition, in this paper we solve optimization problems, including how to trim useless sources, and how to test query containment. By using these techniques, we can generate efficient programs to answer queries.

By taking the query-centric approach, [Li et al. 1998] showed how to generate an executable plan of a query based on source restrictions. If the complete answer to the query cannot be retrieved, [Li et al. 1998] would not answer the query, but would claim that an executable plan does not exist. In this case, our approach can still compute a partial answer. Although we take the query-centric approach in this study, our techniques for finding relevant sources are also applicable to the source-centric approach, since when source views are the same as global predicates, the query-centric approach in [Duschka and Levy 1997] and our framework generate equivalent datalog programs. Other studies on answering queries in the presence of binding restrictions include: how to optimize conjunctive queries [Florescu et al. 1999; Yerneni et al. 1999], how to test whether the complete answer to a query can be computed [Li and Chang 2001], how to describe source capabilities using a powerful language [Vassalos and Papakonstantinou 1997], how to compute mediator capabilities given source capabilities [Yerneni et al. 1999], and how to convert data at mediators [Cluet et al. 1998].

Our containment problem is also called *relative containment* in a recent study [Millstein et al. 2000]. That is, containment is considered with respect to the available data sources. In Section 6 we use a different technique to prove the decidability of relative query containment by taking the query-centric approach. Our technique can also be generalized to the source-centric approach [Li and Chang 1999]. Section 6 shows the differences between our approach and the approach in [Millstein et al. 2000].

2. A MOTIVATING EXAMPLE AND PRELIMINARIES

This section presents our motivating example and introduces the notation used throughout the paper.

EXAMPLE 2. *Assume that we are building a system that integrates the information from four sources of musical CDs, as shown in Table 1. Sources S_1 and S_2 have information about CDs and their songs; sources S_3 and S_4 have information about CDs, their artists, and their prices. To simplify the notation, we use attribute*

Song for song title and attribute Cd for CD title. The "Must Bind" column in the table indicates the attributes that must be specified at a source. For instance, every query sent to S_2 must provide a CD title. In other words, without the information about CD titles, source S_2 cannot be queried to produce answers.

Table 1. Four sources of musical CDs

Source	Contents	Must Bind
S_1	$v_1(Song, Cd)$	$Song$
S_2	$v_2(Song, Cd)$	Cd
S_3	$v_3(Cd, Artist, Price)$	Cd
S_4	$v_4(Cd, Artist, Price)$	$Artist$

Source-view schemas can be represented by a hypergraph [Ullman 1989], in which each node is an attribute and each hyperedge is a source view. The hypergraph of the four views is shown in Figure 1, which also shows the tuples at each source. To simplify the presentation, we use symbols t_i, c_j, and a_k to represent a song title, CD, and artist, respectively. For instance, the source view $v_1(Song, Cd)$ contains two tuples: $\langle t_1, c_1 \rangle$ and $\langle t_2, c_3 \rangle$. The figure shows the adornments of the attributes in each view: b means that the attribute must be bound, f means the attribute can be free.

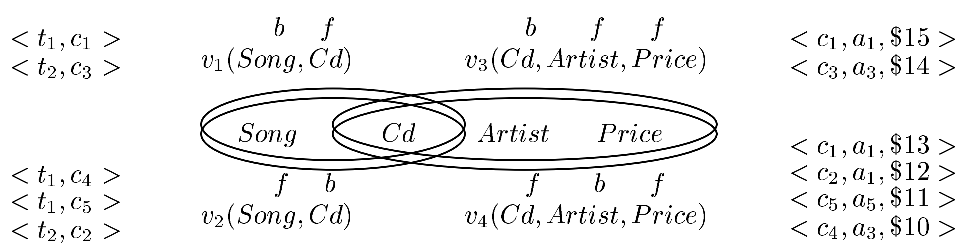

Fig. 1. The hypergraph representation.

Suppose a user wants to find the prices of the CDs that contain a song titled t_1. The answer can be obtained by taking the union of the following four joins: $v_1 \bowtie v_3$, $v_1 \bowtie v_4$, $v_2 \bowtie v_3$, and $v_2 \bowtie v_4$, and performing a selection $Song = t_1$ and then a projection onto the attribute Price. Figure 1 shows that there are four CDs containing the song: $\langle c_1, a_1, \$15 \rangle$, $\langle c_1, a_1, \$13 \rangle$, $\langle c_5, a_5, \$11 \rangle$, and $\langle c_4, a_3, \$10 \rangle$. Therefore, without considering the source restrictions, the answer is $\{\$15, \$13, \$11, \$10\}$. However, due to the limited source capabilities, only the \$15 can be computed if we process each join in the query at one time (as in [Haas et al. 1997; Levy et al. 1996; Li et al. 1998]). The reason is that, $v_1 \bowtie v_3$ yields the \$15 in the answer; $v_1 \bowtie v_4$ cannot be executed by using only v_1 and v_4, since v_4 requires that attribute Artist be specified, but we cannot bind this attribute using only these two views. Similarly, neither of the other two joins can be executed. As a consequence, the user misses the cheaper source for CD c_1 and entirely misses CDs c_4 and c_5.

In this study, we propose a framework that can retrieve more results from sources with restrictions. Instead of considering each join individually, our framework involves other sources not in a join to produce bindings to answer the join. For instance, when joining v_1 and v_4, we also consider the information provided by v_2 and v_3. As we will see in Section 3.3, our framework can find two additional CDs containing the song titled t_1: $\langle c_1, a_1, \$13\rangle$ and $\langle c_4, a_3, \$10\rangle$. If the user wants to find the cheapest CD, our approach can save \$5 for the user!

2.1 Source views

Now we give the notation used throughout the paper. Let an information-integration system have n sources, say, $S_1, \ldots, S_n$. Assume that each source S_i provides its data in the form of a relational view v_i. If sources have other data models, we can use *wrappers* [Hammer et al. 1997] to create a simple relational view of data. In the case where one source has multiple relations, we can represent this source with multiple logical sources, each of which exports only one relational view.

We assume that differences in ontologies, vocabularies, and formats used by sources have been resolved. In particular, if two sources share an attribute name, we assume that the attributes are equivalent, i.e., wrappers take care of any differences. Related research [Maluf and Wiederhold 1997; Papakonstantinou et al. 1995] suggests ways to deal with ontology and format differences. We assume that the schemas of the source views are defined on a global set of attributes. Each view schema is a list of global attributes, and different views may share the same schema. For instance, in Example 2, we have four global attributes: $Song$, $Title$, $Artist$, and $Price$; views v_1 and v_2 share the same schema $(Song, Cd)$.

The query capability of each source is described as a template with a binding pattern [Ullman 1989] representing the possible query forms that the source can accept. The adornments for the attributes in the binding pattern include *b* (the attribute must be bound) and *f* (the attribute can be free). For example, we can use binding pattern *bf* for source view v_1 to describe the fact that every query to v_1 must provide a song title. Similarly, we use binding pattern *fbf* for v_4 to describe the restriction that each query to v_4 must provide an artist name.

For simplicity of exposition, we assume that each view has one template. We use v_i to stand for both the source view and its adorned template, and we believe the distinction should be clear in context. Let $\mathcal{A}(v_i)$ denote the attributes in a source view v_i, and let $\mathcal{B}(v_i)$ and $\mathcal{F}(v_i)$ be the sets of bound and free attributes in the adorned template of v_i, respectively. For instance, in Example 2, $\mathcal{B}(v_1) = \{Song\}$, $\mathcal{F}(v_1) = \{Cd\}$, and $\mathcal{A}(v_1) = \{Song, Cd\}$. Let $\mathcal{V}$ denote the source views with their adornments, $\mathcal{A}(\mathcal{V})$ be the attributes in $\mathcal{V}$.

2.2 Queries

A user query is represented in the form

$$\mathcal{Q} = \langle \mathcal{I}, O, \mathcal{C} \rangle$$

where $\mathcal{I}$ is a list of input assignments of the form $attribute = constant$, O is a list of output attributes whose values the user is interested in, and $\mathcal{C}$ is a list of *connections*. Each connection is a set of source views that connect the input attributes and the output attributes. As we will see shortly, we interpret a connection as the natural join of the views in the connection. The following are some possible ways in which $\mathcal{C}$ could be generated:

(1) It is generated by query expansion at a mediator, as in TSIMMIS.
(2) It is generated by a minimal-connection algorithm, as in universal-relation systems.
(3) It is explicitly specified by the user.

For instance, the query in Example 2 can be represented as

$$\mathcal{Q} = \langle \{Song = t_1\}, \{Price\}, \{T_1, T_2, T_3, T_4\} \rangle$$

in which the four connections are: $T_1 = \{v_1, v_3\}$, $T_2 = \{v_1, v_4\}$, $T_3 = \{v_2, v_3\}$, and $T_4 = \{v_2, v_4\}$. Note that there can be multiple input attributes and multiple output attributes in a query. Let $I(\mathcal{Q})$ and $O(\mathcal{Q})$ respectively denote the input attributes and the output attributes of query $\mathcal{Q}$. $I(\mathcal{Q})$ and $O(\mathcal{Q})$ do not overlap. Let $\mathcal{A}(T)$ be all the attributes in a connection T.

2.3 The answer to a query

Suppose T is a connection in query $\mathcal{Q}$. For those tuples in the *natural join* of the relations in T that satisfy the input constraints in $\mathcal{Q}$, their projections onto the output attributes are the *complete answer to connection* T. The union of the answers to all the connections in $\mathcal{Q}$ is the *complete answer to query* $\mathcal{Q}$. Due to the limited source capabilities, the *obtainable answer to a connection* is the *maximal* answer to the connection that can be retrieved from the sources, using only the initial bindings in the query and the source relations. The union of the obtainable answers to all the connections in $\mathcal{Q}$ is the *obtainable answer to query* $\mathcal{Q}$.

The complete answer to a user query could be retrieved if the sources did not have limited capabilities. However, we may get only a partial answer to the query due to the source restrictions. For instance, in Example 2, the complete answer to the query is $\{\$15, \$13, \$11, \$10\}$, while as we will see in Section 3.3, the obtainable answer is $\{\$15, \$13, \$10\}$. Given source descriptions and a query, if the complete answer to the query cannot be computed, our framework collects as much information as possible to answer the query. In the rest of this paper, unless otherwise specified, the *answer to a connection* means the obtainable answer to the connection, and the *answer to a query* is the union of the obtainable answers to all the connections in the query.

3. A QUERY-PLANNING FRAMEWORK

In this section, we propose a query-planning framework in the presence of source restrictions. In the framework, source descriptions and a query are translated into a datalog program, which can be evaluated to answer the query. The idea of using domain predicates in the framework is borrowed from [Duschka and Levy 1997]. However, in our framework, different domains have different domain predicates, while in [Duschka and Levy 1997] only one domain predicate is used for all attributes. In addition, we use the query-centric approach to information integration, while [Duschka and Levy 1997] uses the source-centric approach to information integration [Duschka 1998].

3.1 Constructing the program $\Pi(\mathcal{Q}, \mathcal{V})$

Given source descriptions $\mathcal{V}$ and a query $\mathcal{Q}$, we translate them into a datalog program, denoted $\Pi(\mathcal{Q}, \mathcal{V})$, that can be evaluated on the source relations. For instance, Figure 2 shows the datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ for the query and the source views in Example 2. We use names beginning with lower-case letters for constants and predicate names, and names beginning with upper-case letters for variables. Note that this program is recursive, although query $\mathcal{Q}$ is not.

r_1:	$ans(P)$	:- $\widehat{v_1}(t_1, C), \widehat{v_3}(C, A, P)$	r_9:	$\widehat{v_3}(C, A, P)$	:- $cd(C), v_3(C, A, P)$
r_2:	$ans(P)$	:- $\widehat{v_1}(t_1, C), \widehat{v_4}(C, A, P)$	r_{10}:	$price(P)$	:- $cd(C), v_3(C, A, P)$
r_3:	$ans(P)$	:- $\widehat{v_2}(t_1, C), \widehat{v_3}(C, A, P)$	r_{11}:	$artist(A)$	:- $cd(C), v_3(C, A, P)$
r_4:	$ans(P)$	:- $\widehat{v_2}(t_1, C), \widehat{v_4}(C, A, P)$	r_{12}:	$\widehat{v_4}(C, A, P)$	:- $artist(A), v_4(C, A, P)$
r_5:	$\widehat{v_1}(S, C)$	:- $song(S), v_1(S, C)$	r_{13}:	$cd(C)$	:- $artist(A), v_4(C, A, P)$
r_6:	$cd(C)$	:- $song(S), v_1(S, C)$	r_{14}:	$price(P)$	:- $artist(A), v_4(C, A, P)$
r_7:	$\widehat{v_2}(S, C)$	:- $cd(C), v_2(S, C)$	r_{15}:	$song(t_1)$	:-
r_8:	$song(S)$	:- $cd(C), v_2(S, C)$			

Fig. 2. The datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ in Example 2.

Let us look at the details of how the program $\Pi(\mathcal{Q}, \mathcal{V})$ is constructed. For each source view v_i, we introduce an EDB predicate v_i and an IDB predicate $\widehat{v_i}$ (called the *α-predicate* of v_i). Predicate v_i represents all the tuples at source S_i, and $\widehat{v_i}$ represents the *obtainable* tuples at S_i. Introduce a goal predicate *ans* to store the answer to the query; the arguments of *ans* correspond to the output attributes $O(\mathcal{Q})$ in $\mathcal{Q}$.

Let $T = \{v_1, \ldots, v_k\}$ be a connection in $\mathcal{Q}$. The following rule is the *connection rule* of T:

$$ans(O(\mathcal{Q})) \text{ :- } \widehat{v_1}(\mathcal{A}(v_1)), \ldots, \widehat{v_k}(\mathcal{A}(v_k))$$

where the arguments in predicate *ans* are the corresponding attributes in $O(\mathcal{Q})$. For each argument in $\widehat{v_i}$, if the corresponding attribute in view v_i is an input attribute of $\mathcal{Q}$, this argument is replaced by the initial value of the attribute in $\mathcal{Q}$. Otherwise, a variable corresponding to the attribute name is used as an argument in predicate $\widehat{v_i}$. For instance, in Figure 2, rules r_1, r_2, r_3, and r_4 are the connection rules of the connections T_1, T_2, T_3, and T_4, respectively.

Decide the domains of all the attributes in the views, and group the attributes into sets while the attributes in each set share the same domain. Introduce a unary *domain predicate* for each domain to represent all its possible values that can be deduced. In Figure 2, the predicates *song*, *cd*, *artist*, and *price* represent the domains of song titles, CD titles, artists, and prices, respectively.

Suppose that source view v_i has m attributes, say $A_1, \ldots, A_m$. Assume the adornment of v_i says that the arguments in positions $1, \ldots, p$ need to be bound, and the arguments in positions $p+1, \ldots, m$ can be free. The following rule is the *α-rule* of v_i:

$$\widehat{v_i}(A_1, \ldots, A_m) \text{ :- } domA_1(A_1), \ldots, domA_p(A_p), v_i(A_1, \ldots, A_m)$$

in which each $domA_j$ ($j = 1, \ldots, p$) is the domain predicate for attribute A_j. For

$k = p+1, \ldots, m$, the following rule is a *domain rule* of v_i:

$$domA_k(A_k) \text{ :- } domA_1(A_1), \ldots, domA_p(A_p), v_i(A_1, \ldots, A_m)$$

For instance, rule r_9 in Figure 2 is the α-rule of v_3; rules r_{10} and r_{11} are its domain rules. Assume that $A_i = a_i$ is in the assignment list $\mathcal{I}$ of $\mathcal{Q}$, the following rule is a *fact rule* of attribute A_i:

$$domA_i(a_i) \text{ :- }$$

For instance, rule r_{15} in Figure 2 is a fact rule of attribute *Song*, since we know from the query that t_1 is a song title.

The program $\Pi(\mathcal{Q}, \mathcal{V})$ is constructed in three steps:

(1) Write the connection rule for each connection in $\mathcal{Q}$.

(2) Write the α-rule and the domain rules for each source view in $\mathcal{V}$.

(3) Write the fact rule for each input attribute in $\mathcal{Q}$.

In Figure 2, rules r_1, r_2, r_3, and r_4 are the connection rules of T_1, T_2, T_3, and T_4, respectively. Rule r_5 is the α-rule of v_1, and r_6 is the domain rule of v_1; rules r_7 to r_{14} are the α-rules and the domain rules of the other three source views. Finally, r_{15} is the fact rule of the attribute *Song*. Recall that the views in each connection link the input attributes and the output attributes in the query. Based on how program $\Pi(\mathcal{Q}, \mathcal{V})$ is constructed, we have the following proposition:

PROPOSITION 1. *Given source descriptions $\mathcal{V}$ and a query $\mathcal{Q}$, the datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ is safe.*

3.2 Binding assumptions

During the construction of the program $\Pi(\mathcal{Q}, \mathcal{V})$, we make the following important assumptions:

(1) Each binding for an attribute must be from the domain of this attribute.

(2) If a source view requires a value, say, a string, as a particular argument, we will not allow the strategy of trying all the possible strings to "test" the source.

(3) Rather we assume that any binding is either obtained from the user query, or from a tuple returned by another source query.

We use Example 2 to explain these assumptions. The first assumption says that we would not use an artist name as a binding for attribute *Song*. View $v_3(Cd, Artist, Price)$ requires each query to source S_3 to give a CD title. The second assumption says that we would *not* allow the following naive "strategy": generate all possible strings to test whether S_3 has CDs with these strings as titles. This approach would not terminate, since there will be an infinite number of strings that need to be tested. The third assumption says that each binding of an attribute A must either be derived from the user query, or be a value of A in a tuple returned by another source query. For instance, if c_1 is a CD title returned from source S_1, and $Cd = c_2$ is an initial binding in a query, then we know that c_1 and c_2 are two CD titles, and we can use them to query source S_3. In Section 7 we will discuss other possibilities for obtaining bindings.

3.3 Evaluating the program $\Pi(\mathcal{Q}, \mathcal{V})$

We evaluate the datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ on the source relations to compute the facts for predicate *ans*. Note that the v_i's are the only EDB predicates in $\Pi(\mathcal{Q}, \mathcal{V})$. However, because of the source restrictions, we do not know the tuples at each source before sending source queries. Now we show how to evaluate $\Pi(\mathcal{Q}, \mathcal{V})$ to answer the query.

To evaluate the domain rules and the α-rule of a source view v_i, predicate v_i is "populated" by source queries to S_i. Suppose that the right-hand side of its domain rules and its α-rule is:

$$domA_1(A_1), \ldots, domA_p(A_p), v_i(A_1, \ldots, A_m)$$

Once we know that $(a_1, \ldots, a_p)$ are the values of the $domA_j$'s ($j = 1, \ldots, p$), respectively, we can send a query $v_i(a_1, \ldots, a_p, A_{p+1}, \ldots, A_m)$ to source S_i. This source query is guaranteed to be executable, since it satisfies the binding requirements of v_i. The results of this source query add more tuples to the predicate $\widehat{v_i}$ (for the α-rule) and the predicates $domA_j$'s (for the domain rules).

After the evaluation of the program terminates, the facts for the domain predicates include *all* the obtainable values of these domains. Similarly, the α-predicate facts are *all* the obtainable tuples at the sources. Thus, we have the following proposition:

PROPOSITION 2. *Given source descriptions $\mathcal{V}$ and a query $\mathcal{Q}$, for any database $\mathcal{D}$ of $\mathcal{V}$, if we evaluate $\Pi(\mathcal{Q}, \mathcal{V})$ on $\mathcal{D}$, the facts for the predicate ans are the obtainable answer to $\mathcal{Q}$.*

Table 2. Evaluating the program in Figure 2.

Order	Source Query	Returned Tuple(s)	New Bindings(s)
1	$v_1(t_1, C)$	$\langle t_1, c_1\rangle$	$Cd = c_1$
2	$v_3(c_1, A, P)$	$\langle c_1, a_1, \$15\rangle$	$Artist = a_1$
3	$v_4(C, a_1, P)$	$\langle c_1, a_1, \$13\rangle$,$\langle c_2, a_1, \$12\rangle$	$Cd = c_2$
4	$v_2(S, c_2)$	$\langle t_2, c_2\rangle$	$Song = t_2$
5	$v_1(t_2, C)$	$\langle t_2, c_3\rangle$	$Cd = c_3$
6	$v_3(c_3, A, P)$	$\langle c_3, a_3, \$14\rangle$	$Artist = a_3$
7	$v_4(C, a_3, P)$	$\langle c_4, a_3, \$10\rangle$	$Cd = c_4$
8	$v_2(S, c_4)$	$\langle t_1, c_4\rangle$	

Table 3. Results

IDBs	**Results**	**IDBs**	**Results**
$\widehat{v_1}$	$\langle t_1, c_1\rangle\langle t_2, c_3\rangle$	*song*	t_1, t_2
$\widehat{v_2}$	$\langle t_1, c_4\rangle\langle t_2, c_2\rangle$	*cd*	c_1, c_2, c_3, c_4
$\widehat{v_3}$	$\langle c_1, a_1, \$15\rangle\langle c_3, a_3, \$14\rangle$	*artist*	a_1, a_3
$\widehat{v_4}$	$\langle c_1, a_1, \$13\rangle\langle c_2, a_1, \$12\rangle\langle c_4, a_3, \$10\rangle$	*price*	$\$15, \$14, \$13, \$12, \$10$
ans	$\$15, \$13, \$10$		

Table 2 shows how to evaluate the program in Figure 2 to compute the answer to the query in Example 2, and Table 3 shows the results. Clearly the program computes all the obtainable values of song titles, CD titles, artists, and prices from the four sources and the query, as well as all the obtainable tuples at the sources. The set of *ans* facts is the answer to the query. Therefore, our approach returns two more tuples, \$13 and \$10, than the approach in Example 2. Note that we cannot retrieve the tuple $\langle t_1, c_5\rangle$ of v_2 or the tuple $\langle c_5, a_5, \$11\rangle$ of v_4, since we cannot obtain the binding a_5 for attribute *Artist*, no matter what legal source queries we execute.

The program $\Pi(\mathcal{Q}, \mathcal{V})$ is constructed in a brute-force way, and it needs to be optimized. In particular, for each connection T in the query, the program may access views that are not in T. Some of these off-connection accesses do not add anything to the query's answer. We thus want to include judiciously only those sources that provide some values at a place where they are needed. In the rest of the paper, we solve some optimization problems in this framework, including how to decide whether accessing off-query sources is necessary, how to choose the relevant sources to obtain useful bindings, and how to test query containment.

4. WHEN ACCESSING OFF-CONNECTION SOURCES IS NECESSARY

In this section, we discuss in what cases accessing off-connection views is necessary to answer a connection. The following example shows that accessing off-connection views is not always necessary.

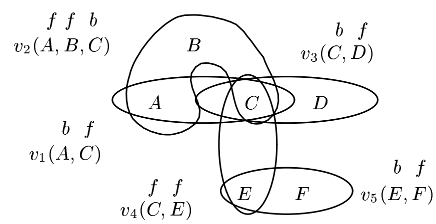

Fig. 3. Source views in Example 3.

EXAMPLE 3. *Consider the five views in Figure 3. Suppose that a user submits a query*

$$\mathcal{Q} = \langle \{A = a_0\}, \{D\}, \{T_1, T_2\}\rangle,$$

which has two connections $T_1 = \{v_1, v_3\}$, $T_2 = \{v_2, v_3\}$. That is, the user knows the value of A is a_0, and wants to get the associated D values using $v_1 \bowtie v_3$ and $v_2 \bowtie v_3$. Assume that different attributes have different domains. The corresponding datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ is shown in Figure 4.

Consider connection T_1. The program $\Pi(\mathcal{Q}, \mathcal{V})$ accesses the three views that are not in T_1 during the evaluation of the program. However, these off-connection accesses do not contribute to T_1's results. Indeed, suppose $t = \langle d\rangle$ is a tuple in the complete answer to T_1, and t comes from tuple $t_1 = \langle a_0, c\rangle$ of v_1 and tuple $t_3 = \langle c, d\rangle$ of v_3. By sending a query $v_1(a_0, C)$ to S_1 we can retrieve tuple t_1. With the new binding $C = c$, we can send a query $v_3(c, D)$ to S_3, and retrieve tuple t_3. Therefore, by using only the views in T_1 we can compute its complete answer.

r_1:	$ans(D)$	:- $\widehat{v_1}(a_0, C), \widehat{v_3}(C, D)$	r_9:	$domD(D)$	:- $domC(C), v_3(C, D)$
r_2:	$ans(D)$	:- $\widehat{v_2}(a_0, B, C), \widehat{v_3}(C, D)$	r_{10}:	$\widehat{v_4}(C, E)$	:- $v_4(C, E)$
r_3:	$\widehat{v_1}(A, C)$	:- $domA(A), v_1(A, C)$	r_{11}:	$domC(C)$	:- $v_4(C, E)$
r_4:	$domC(C)$	:- $domA(A), v_1(A, C)$	r_{12}:	$domE(E)$	:- $v_4(C, E)$
r_5:	$\widehat{v_2}(A, B, C)$	:- $domC(C), v_2(A, B, C)$	r_{13}:	$\widehat{v_5}(E, F)$	:- $domE(E), v_5(E, F)$
r_6:	$domA(A)$	:- $domC(C), v_2(A, B, C)$	r_{14}:	$domF(F)$	:- $domE(E), v_5(E, F)$
r_7:	$domB(B)$	:- $domC(C), v_2(A, B, C)$	r_{15}:	$domA(a_0)$	:-
r_8:	$\widehat{v_3}(C, D)$	:- $domC(C), v_3(C, D)$			

Fig. 4. The datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ in Example 3.

Consider connection T_2. Since we cannot get any binding for attribute C by using only the two views in T_2, we need v_2 and v_4 to contribute C bindings. Thus these two off-connection views are useful to T_2. On the other hand, $v_5(E, F)$ does not contribute to T_2's results, because the E and F bindings from S_5 do not help obtain more answers to T_2.

In general, given a connection T in a query $\mathcal{Q}$, we need to decide whether accessing the views outside T is necessary. Before giving the solution, we first introduce some notation.

4.1 Forward-closure

Definition 1. *Given a set of source views $\mathcal{W} \subseteq \mathcal{V}$ and a set of attributes $X \subseteq \mathcal{A}(\mathcal{V})$, the forward-closure of X given $\mathcal{W}$, denoted f-closure$(X, \mathcal{W})$, is a set of the source views in $\mathcal{W}$ such that, starting from the attributes in X as the initial bindings, the binding requirements of these source views are satisfied by using only the source views in $\mathcal{W}$.*

f-closure$(X, \mathcal{W})$ can be computed as follows. At the beginning, only the attributes in X are bound, and *f-closure*$(X, \mathcal{W})$ is empty. At each step, for each source view $v \in \mathcal{W} -$ *f-closure*$(X, \mathcal{W})$, check whether $\mathcal{B}(v)$, the bound attributes of v, is a subset of the bound attributes so far. If so, add v to *f-closure*$(X, \mathcal{W})$, and each attribute in $\mathcal{F}(v)$, the free attributes of v, becomes bound. Repeat this process until no more source views can be added to *f-closure*$(X, \mathcal{W})$. Let $\mathcal{A}($*f-closure*$(X, \mathcal{W}))$ denote all the attributes of the source views in *f-closure*$(X, \mathcal{W})$. Therefore, $\mathcal{A}($*f-closure*$(X, \mathcal{W}))$ includes all the attributes that can be bound eventually by using the source views in $\mathcal{W}$ starting from the initial bindings in X.

EXAMPLE 4. *In Example 3, f-closure$(\{A\}, \{v_1, v_2, v_3\}) = \{v_1, v_2, v_3\}$, since we can use the bound attribute A to get tuples of v_1 and bind C, which is the only bound attribute of v_2 and v_3. In Example 2, f-closure$(\{Song\}, \{v_1, v_4\}) = \{v_1\}$, and f-closure$(\{Song\}, \{v_1, v_3\}) = \{v_1, v_3\}$.*

4.2 Independent connections

Definition 2. *A connection T in a query $\mathcal{Q}$ is independent if*

$$f\text{-}closure(I(\mathcal{Q}), T) = T.$$

That is, the binding requirements of the source views in the connection can be satisfied by using only these source views and starting from the initial bindings in $I(\mathcal{Q})$.

In other words, if connection $T = \{w_1, \ldots, w_k\}$ is independent, then there exists an *executable sequence* of all the source views in connection T: $w_{i_1}, \ldots, w_{i_k}$, such that $\mathcal{B}(w_{i_1}) \subseteq I(\mathcal{Q})$, and for $j = 2, \ldots, k$, $\mathcal{B}(w_{i_j}) \subseteq I(\mathcal{Q}) \cup \mathcal{A}(w_{i_1}) \cup \cdots \cup \mathcal{A}(w_{i_{j-1}})$. For instance, the connection $T_1 = \{v_1, v_3\}$ in Example 3 is independent, since it has an executable sequence: v_1, v_3. The following theorem shows that an independent connection does not require bindings from views outside the connection.

THEOREM 1. *If connection T is independent, then for any database of the sources, we can compute the complete answer to T by using only the source views in T.*

PROOF. Suppose connection T has k source views, and it has an executable sequence $v_1, \ldots, v_k$. Consider each tuple t in the complete answer to T. Assume that tuple t comes from tuples $t_1, \ldots, t_k$ of source views $v_1, \ldots, v_k$, respectively. Since $v_1, \ldots, v_k$ is an executable sequence, the binding requirements of v_1 are satisfied by $I(\mathcal{Q})$, i.e., $\mathcal{B}(v_1) \subseteq I(\mathcal{Q})$. Thus, we can send a source query to S_1 by binding the attributes in $\mathcal{B}(v_1)$ to their initial values in $\mathcal{Q}$, and retrieve the tuple t_1 from v_1. We then use the bound values of $I(\mathcal{Q}) \cup \mathcal{F}(v_1)$ to send S_2 a source query to get tuple t_2. Repeat this process following the executable sequence, until we retrieve all the t_i's. Therefore, by using only the views in T, we can retrieve the tuple t in the complete answer to the connection. □

THEOREM 2. *For a nonindependent connection T, there exists a database of the sources, such that some tuples in the complete answer to T cannot be obtained.*

PROOF. If connection T is not independent, i.e., *f-closure*$(I(\mathcal{Q}), T) \neq T$, we construct an instance of source relations, such that a tuple in the complete answer to the connection cannot be obtained. Let $\mathcal{A}(T) = \{A_1, \ldots, A_n\}$ be the set of attributes in T. Let tuple $t = (a_1, \ldots, a_n)$, where a_i is a distinct value for attribute A_i. If A_i is in $I(\mathcal{Q})$, then its value in t, a_i, is its initial value in $\mathcal{Q}$. Each view v_i in T has only one tuple t_i, which is the projection of t onto the attributes $\mathcal{A}(v_i)$. Other sources are empty. Then the projection of t onto $O(\mathcal{Q})$ is in the complete answer to T. However, since *f-closure*$(I(\mathcal{Q}), T) \neq T$, and all other sources are empty, we cannot get the necessary bindings to retrieve all the tuples t_i's, and we cannot compute any answer to connection T. □

Many related studies (e.g., [Florescu et al. 1999; Li et al. 1998]) consider the case where a connection in a query is independent. If the connection is not independent, their algorithms give up attempting to answer the connection. However, our framework can still compute a partial answer to the connection by accessing off-connection views.

5. FINDING RELEVANT SOURCE VIEWS

For a nonindependent connection, not all its off-connection accesses can contribute to the connection's results. In this section, we discuss what sources should be accessed to answer a connection. To simplify the presentation, in the rest of the paper we assume that different attributes are from different domains.

DEFINITION 3. *Given source descriptions $\mathcal{V}$, a query $\mathcal{Q}$, and a connection T in $\mathcal{Q}$, a source view $v \in \mathcal{V}$ is* relevant *to connection T if for some source relations, removing v from $\mathcal{V}$ can change the obtainable answer to connection T; otherwise, v is* irrelevant *to connection T.*

In other words, a source view is relevant to a connection T if we can miss some answers to T if we do not use this view. Note that whether a source view is relevant to a connection does not depend on other connections in the query.

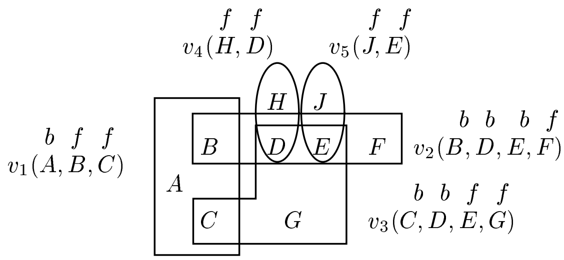

Fig. 5. The source views in Example 5.

EXAMPLE 5. *Consider the five views in Figure 5. Suppose that a user submits a query*

$$\mathcal{Q} = \langle\{A = a\}, \{F, G\}, \{T\}\rangle,$$

which has one connection $T = \{v_1, v_2, v_3\}$. That is, the user knows the value of A is a, and wants to get the associated F and G values using $v_1 \bowtie v_2 \bowtie v_3$. Connection T is not independent, since we cannot bind attributes D and E by using only the views in T starting from the initial binding in $\mathcal{Q}$. We need other views to bind D and E, so that we can query S_2 and S_3 to retrieve tuples. Thus, v_4 and v_5 may be useful.

However, although view v_5 can bind attribute E, it is not relevant to connection T. To illustrate the reason, we prove that the obtainable answers to T can be computed by using only v_1, v_2, v_3, and v_4. Suppose tuple $t = \langle f, g\rangle$ is in the obtainable answer, and t comes from tuple $t_1 = \langle a, b, c\rangle$ of v_1, tuple $t_2 = \langle b, d, e, f\rangle$ of v_2, and tuple $t_3 = \langle c, d, e, g\rangle$ of v_3. Since the initial value of A in the query is a, we can send a source query $v_1(a, B, C)$ to retrieve tuple t_1 from v_1. Because attribute D is not in $I(\mathcal{Q})$, and only v_4 (with binding pattern ff) takes D as a free attribute, the value d of D must be derived from the result of a source query to S_4, which includes a tuple whose D value is d. With $C = c$ and $D = d$, we can retrieve tuple t_3 from v_3 by sending a source query $v_3(c, d, E, G)$, and then retrieve tuple t_2 from v_2 by sending a source query $v_2(b, d, e, F)$. Thus, without using v_5, we can get tuple t in the obtainable answer to connection T. The proof also shows that without using v_4, we cannot get any answer to T.

As there may be many views with different schemas and binding patterns, it becomes challenging to decide which views can really contribute to the results of a connection. Before giving the algorithm for finding all the relevant views of a connection, we require a series of definitions.

5.1 Queryable source views

A source view is *queryable* if it is in *f-closure*$(I(\mathcal{Q}), \mathcal{V})$. All the queryable source views are those that we may eventually query, starting from the initial bindings in $I(\mathcal{Q})$, and perhaps using several preliminary queries to other sources in order to get the bindings we need for these source views. Let $\mathcal{V}_q$ denote all the queryable source views in $\mathcal{V}$, and $\mathcal{A}(\mathcal{V}_q)$ be all the attributes in $\mathcal{V}_q$.

We cannot get any tuples from a nonqueryable source view, no matter what the source relations are. If a connection contains a nonqueryable source view, we cannot get any answer to this connection. Thus we need to consider only the *queryable connections* in $\mathcal{Q}$, i.e., the connections that do not have any nonqueryable source view. Clearly an independent connection is also a queryable connection, but not vice versa. For instance, in Example 3, connection T_2 is queryable, since both v_2 and v_3 are queryable source views, but T_2 is not independent.

5.2 Kernel, BF-chain, and backward-closure

DEFINITION 4. *Assume T is a queryable connection in query $\mathcal{Q}$. A set of attributes $\mathcal{K} \subseteq \mathcal{A}(T)$ is a kernel of T if*

$$f\text{-}closure\big(\mathcal{K} \cup I(\mathcal{Q}), T\big) = T$$

and

$$f\text{-}closure\big((\mathcal{K} - \{A\}) \cup I(\mathcal{Q}), T\big) \neq T$$

for any attribute $A \in \mathcal{K}$.

Intuitively, a kernel $\mathcal{K}$ of connection T is a minimal set of attributes in $\mathcal{A}(T)$ such that, if the attributes in $\mathcal{K}$ have been bound, together with the initial bindings in $I(\mathcal{Q})$, we can bind all the attributes $\mathcal{A}(T)$ by using only the source views in T. In Example 3, $\{C\}$ is a kernel of connection T_2, because $f\text{-}closure(\{C\} \cup I(\mathcal{Q}), T_2) = f\text{-}closure(\{C, A\}, T_2) = T_2$. In Example 5, $\{D\}$ is a kernel of the connection T, while $\{D, E\}$ is not. Since a kernel of a connection must be minimal, it cannot share any attribute with $I(\mathcal{Q})$.

We compute a kernel of a connection T by shrinking the set of attributes $X = \mathcal{A}(T) - I(\mathcal{Q})$ as much as possible, while X satisfies: $f\text{-}closure(X \cup I(\mathcal{Q}), T) = T$. When X cannot be smaller, it will be a kernel of T. An independent connection has only one kernel: the empty set. A nonindependent connection has only nonempty kernels. It may have multiple kernels, as shown by the following example.

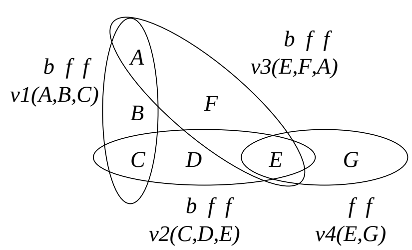

Fig. 6. Multiple kernels of a connection.

EXAMPLE 6. *Figure 6 shows a hypergraph of four source views. The binding patterns for $v_1(A, B, C)$, $v_2(C, D, E)$, and $v_3(E, F, A)$ are all bff, and the binding pattern for $v_4(E, G)$ is ff. Assume a user query is $\mathcal{Q} = \langle\{B = b_0\}, \{A, C, E\}, \{T\}\rangle$, in which the only connection is $T = \{v_1, v_2, v_3\}$. T has three kernels: $\{A\}$,*

$\{C\}$, and $\{E\}$. For instance, $\{A\}$ is a kernel because f-closure$(\{A\} \cup I(\mathcal{Q}), T)$ $=$ f-closure$(\{A, B\}, \{v_1, v_2, v_3\}) = \{v_1, v_2, v_3\} = T$.

DEFINITION 5. *A sequence of queryable source views $w_1, \ldots, w_k$ (i.e., each $w_i \in \mathcal{V}_q$) forms a BF-chain (bound-free chain) if for $i = 1, \ldots, k-1$, $\mathcal{F}(w_i) \cap \mathcal{B}(w_{i+1})$ is not empty. The source views w_1 and w_k are the head and the tail of the BF-chain, respectively.*

In other words, for every two adjacent views in a BF-chain, the free attributes of the first one overlap with the bound attributes of the second, and thus the first view contributes bindings to the second one. In Example 3, (v_4, v_2, v_1, v_3) is a BF-chain, in which v_4 is the head and v_3 is the tail.

DEFINITION 6. *Suppose A is an attribute in $\mathcal{A}(\mathcal{V}_q)$. The backward-closure of A, denoted b-closure(A), is the set of queryable views that can be backtracked from A by following some BF-chain in a reverse order, in which A is a free attribute of the tail in the BF-chain.*

We can compute *b-closure*(A) as follows. Start by setting *b-closure*(A) to those source views in $\mathcal{V}_q$ that take A as a free attribute. For each view $v \in \mathcal{V}_q -$ *b-closure*(A), if there is a view $w \in$ *b-closure*(A) such that $\mathcal{F}(v)$ and $\mathcal{B}(w)$ overlap, then add v to *b-closure*(A). Repeat this process until no more queryable source views can be added to *b-closure*(A). In Example 3, the backward-closure of attribute C is $\{v_1, v_2, v_4\}$. The backward-closure of a set of attributes $X \subseteq \mathcal{A}(\mathcal{V}_q)$, denoted *b-closure*$(X)$, is the union of all the backward-closures of the attributes in X, i.e., *b-closure*$(X) = \bigcup_{A \in X}$ *b-closure*(A). By the definitions of kernel, BF-chain, and backward-closure, we have the following lemmas.

LEMMA 1. *If $\mathcal{K}$ is a kernel of a queryable connection T and A is an attribute in $\mathcal{K}$, then A is not in $\mathcal{A}\Big($f-closure$\big((\mathcal{K} - \{A\}) \cup I(\mathcal{Q}), T\big)\Big)$. That is, starting from the attributes of $(\mathcal{K} - \{A\}) \cup I(\mathcal{Q})$ as the initial bindings, we cannot bind attribute A by using only the source views in T.*

PROOF. Suppose that attribute A is in $\mathcal{K}$, and A is also in $\mathcal{A}\Big($*f-closure*$\big((\mathcal{K} - \{A\}) \cup I(\mathcal{Q}), T\big)\Big)$. Then starting from the attributes of $(\mathcal{K} - \{A\}) \cup I(\mathcal{Q})$ as the initial bindings, we can bind attribute A by using only the source views in T. Therefore, we can bind all the attributes in $\mathcal{K}$, and then bind all the attributes in $\mathcal{A}(T)$ (since $\mathcal{K}$ is a kernel of T). Thus, $\mathcal{K} - \{A\}$ would be a kernel of connection T. Then $\mathcal{K}$ could not be a kernel since it is not minimal. □

LEMMA 2. *If A_1 and A_2 are two attributes, and there is a BF-chain such that A_1 is a bound attribute of the head and A_2 is a free attribute of the tail, then b-closure$(A_1) \subseteq$ b-closure(A_2).*

PROOF. Since A_2 is a free attribute of the BF-chain tail, we can backtrack from A_2 along the BF-chain until we reach A_1 in the head. Thus all the views on the BF-chain are in *b-closure*(A_2). Based on how *b-closure*(A_2) is computed, all the views in *b-closure*(A_1) are also added into *b-closure*(A_2) during the computation of *b-closure*(A_2). Therefore, *b-closure*$(A_1) \subseteq$ *b-closure*(A_2). □

LEMMA 3. *If connection T has two different kernels $\mathcal{K}_1$, $\mathcal{K}_2$, then b-closure*$(\mathcal{K}_1) =$ *b-closure*$(\mathcal{K}_2)$.

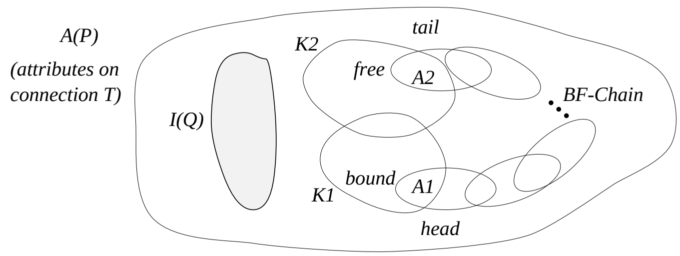

Fig. 7. Proof of Lemma 3.

PROOF. The main idea of the proof is shown in Figure 7. Since connection T has two different kernels, T cannot be independent, and both $\mathcal{K}_1$ and $\mathcal{K}_2$ are not empty. Since $\mathcal{K}_1$ is a kernel of T, for each attribute in $\mathcal{K}_1$, say A_1, by Lemma 1, we have $A_1 \notin \mathcal{A}\Big(\textit{f-closure}\big((\mathcal{K}_1 - \{A_1\}) \cup I(\mathcal{Q}), T\big)\Big)$. We also have $\textit{f-closure}(\mathcal{K}_1 \cup I(\mathcal{Q}), T) = T$, while $\textit{f-closure}((\mathcal{K}_1 - \{A_1\}) \cup I(\mathcal{Q}), T) \neq T$. In addition, $\mathcal{A}(\textit{f-closure}((\mathcal{K}_1 - \{A_1\}) \cup I(\mathcal{Q}), T))$ cannot be a superset of $\mathcal{K}_2$, otherwise starting from the attributes of $(\mathcal{K}_1 - \{A_1\}) \cup I(\mathcal{Q})$ as initial bindings and using only the source views in T, we could bind all the attributes in $\mathcal{K}_2$, and then bind all the attributes in T (since $\mathcal{K}_2$ is a kernel of T), and $\mathcal{K}_1 - \{A_1\}$ would be a kernel.

Let A_2 be an attribute in $\mathcal{K}_2$ that is not in $\mathcal{A}(\textit{f-closure}(\mathcal{K}_1 - \{A_1\}) \cup I(\mathcal{Q}), T))$. As shown in Figure 7, since A_2 must be in $\mathcal{A}(\textit{f-closure}(\mathcal{K}_1) \cup I(\mathcal{Q}), T))$, then either $A_2 = A_1$, or there must exist a BF-chain, such that all the source views on the BF-chain are in T, and the head of the BF-chain takes A_1 as a bound attribute, and the tail takes A_2 as a free attribute. Note that in Figure 7, the attributes in $I(\mathcal{Q})$ may overlap with the attributes on the BF-chain.

If $A_2 = A_1$, then $\textit{b-closure}(A_1) = \textit{b-closure}(A_2) \subseteq \textit{b-closure}(\mathcal{K}_2)$. If $A_2 \neq A_1$, then the above BF-chain exists. By Lemma 2, $\textit{b-closure}(A_1) \subseteq \textit{b-closure}(A_2) \subseteq \textit{b-closure}(\mathcal{K}_2)$. Then we have $\textit{b-closure}(\mathcal{K}_1) \subseteq \textit{b-closure}(\mathcal{K}_2)$, since $\textit{b-closure}(\mathcal{K}_1) = \bigcup_{A \in \mathcal{K}_1}(\textit{b-closure}(A))$. Similarly, we can prove $\textit{b-closure}(\mathcal{K}_2) \subseteq \textit{b-closure}(\mathcal{K}_1)$. Therefore, $\textit{b-closure}(\mathcal{K}_1) = \textit{b-closure}(\mathcal{K}_2)$. □

For instance, in Example 6, the connection $T = \{v_1, v_2, v_3\}$ has three kernels: $\{A\}$, $\{C\}$, and $\{E\}$, and they have the same backward-closure: $\{v_1, v_2, v_3, v_4\}$.

5.3 The algorithm FIND_REL

Now we show how to find all the relevant views of a connection by giving the following theorem:

THEOREM 3. *If $\mathcal{K}$ is a kernel of a queryable connection T, then b-closure$(\mathcal{K}) \cup T$ are all the relevant source views of connection T.*

PROOF. Figure 8 shows the essential idea of the proof. For a kernel $\mathcal{K}$ of a queryable connection T, we can prove: (i) All the source views in $\mathcal{V} - \textit{b-closure}(\mathcal{K}) \cup T$

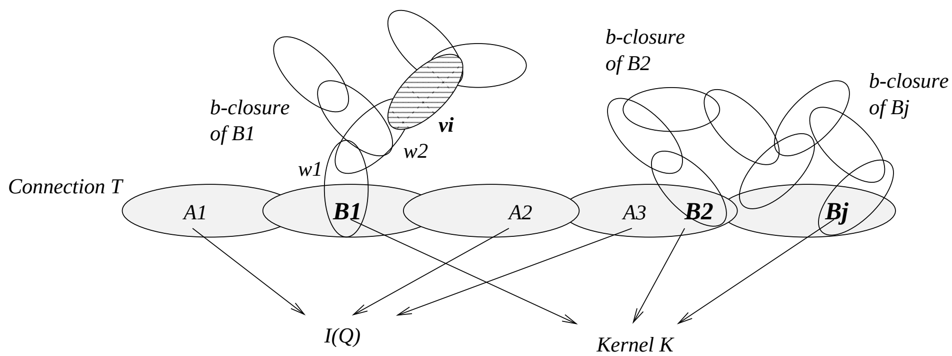

Fig. 8. Relevant source views of a queryable connection.

are irrelevant to T; (ii) Every source view in T is relevant to T; (iii) Every source view in $b\text{-}closure(\mathcal{K})$ is relevant to T. We prove (i) by showing that we can get all the tuples in the obtainable answer to T by using only the source views in $b\text{-}closure(\mathcal{K}) \cup T$. We prove (ii) by constructing a database of the source views, such that the obtainable answer to T is not empty, while if we remove any source view in T, the obtainable answer to T becomes empty. To prove (iii), for every source view v_i in $b\text{-}closure(\mathcal{K})$, we prove that v_i is relevant to T by constructing an instance of the source relations, such that without using v_i, we will miss a tuple in the obtainable answer to T. □

Note that the backward-closures of different attributes in the kernel may overlap, and they may also overlap with the source views in T. In addition, if T is independent, then it has only one kernel, the empty set, whose backward-closure is empty. Thus only the source views in T are relevant to T, and this claim is consistent with Theorem 1.

Using Theorem 3, we give the algorithm FIND_REL that finds all the relevant source views of a queryable connection in a query. The algorithm is shown in Figure 9.

Algorithm FIND_REL: Find the relevant views of a connection.
Input: • $\mathcal{V}$: Source views with binding restrictions.
• $\mathcal{Q}$: A query.
• T: A queryable connection in $\mathcal{Q}$.
Output: All the relevant views of T.
Method:
(1) Compute all the queryable source views $\mathcal{V}_q = f\text{-}closure(I(\mathcal{Q}), \mathcal{V})$.
(2) Compute a kernel $\mathcal{K}$ of connection T.
(3) Compute the backward-closure $b\text{-}closure(\mathcal{K})$.
(4) Return $b\text{-}closure(\mathcal{K}) \cup T$.

Fig. 9. The algorithm FIND_REL.

EXAMPLE 7. *In Example 3, all the five source views are queryable. Connection* $T_1 = \{v_1, v_3\}$ *is independent, so the only relevant source views of* T_1 *are* v_1 *and* v_3*. Connection* $T_2 = \{v_2, v_3\}$ *is not independent, and it has only one kernel:*

$\{C\}$. The backward-closure of the kernel is $\{v_1, v_2, v_4\}$, so only v_1, v_2, v_3, and v_4 are relevant to T_2.

In Example 5, connection $T = \{v_1, v_2, v_3\}$ has one kernel $\{D\}$, whose backward-closure is $\{v_4\}$. Thus the relevant source views of the connection are v_1, v_2, v_3, and v_4. The connection in Example 6 has three kernels: $\{A\}$, $\{C\}$, and $\{E\}$. We choose one of them, say $\{A\}$, and compute its backward-closure, which is $\{v_1, v_2, v_3, v_4\}$. Thus all the four views are relevant to the connection.

Let us analyze the complexity of the algorithm FIND_REL. Suppose that there are n source views in $\mathcal{V}$. Consider a queryable connection T with m source views and k attributes. Assume it takes $O(1)$ time to check whether a set of attributes is a subset of another set of attributes. As described in Section 5.1, we can get all the queryable source views by computing *f-closure*$(I(\mathcal{Q}), \mathcal{V})$. Step 1 thus can be done in $O(n^2)$ time. Step 2 can be done by following the approach described in Section 5.2, which shrinks the attributes in $\mathcal{A}(T) - I(\mathcal{Q})$ as much as possible. Since for each set of attributes $X \subseteq \mathcal{A}(T) - I(\mathcal{Q})$, it takes $O(m^2)$ time to compute *f-closure*$(X \cup I(\mathcal{Q}), T)$, step 2 can be done in $O(km^2)$ time.

In step 3, for each attribute A in a kernel $\mathcal{K}$ of T, *b-closure*(A) can be computed in $O(n^2)$ time because during the computation, we can keep a set of attributes $\mathcal{A}_b$ as the union of the $\mathcal{B}(w_i)$'s for each w_i in *b-closure*(A) that has been computed so far. At each step, for each queryable source view v that is not in the current *b-closure*(A), we check whether $\mathcal{F}(v) \cap \mathcal{A}_b$ is not empty. If so, v is added to *b-closure*(A). Thus step 3 can be done in $O(kn^2)$ time. Therefore, the total time complexity of finding the relevant source views of the connection is $O(n^2)+O(km^2)+O(kn^2) = O(k(m^2+n^2))$ $= O(kn^2)$.

5.4 Constructing an efficient program

Given source descriptions $\mathcal{V}$ and a query $\mathcal{Q}$, we can construct an efficient program using Theorem 3. We first find the relevant views of all the connections in $\mathcal{Q}$ as follows:

(1) Compute all the queryable source views $\mathcal{V}_q =$ *f-closure*$(I(\mathcal{Q}), \mathcal{V})$.

(2) Remove the nonqueryable connections, i.e., the connections that have a nonqueryable view.

(3) Compute the relevant views for each queryable connection by calling the algorithm FIND_REL.

(4) Take the union of all these relevant source views.

We then use only these relevant source views (denoted $\mathcal{V}_r$) of query $\mathcal{Q}$ to construct a datalog program $\Pi(\mathcal{Q}, \mathcal{V}_r)$ in the same way as $\Pi(\mathcal{Q}, \mathcal{V})$ is constructed. For instance, in Example 3, all the five source views are queryable. By calling the algorithm FIND_REL we find that views v_1 and v_3 are relevant to connection T_1; views v_1, v_2, v_3, and v_4 are relevant to connection T_2. Therefore, the relevant views for both connections are v_1, v_2, v_3, and v_4. We use these four views to construct a more efficient program, which can be obtained by dropping the rules r_{13} and r_{14} in Figure 4.

In addition, some useless rules in the program $\Pi(\mathcal{Q}, \mathcal{V}_r)$ can be removed, since they do not contribute to the answer. For instance, in Example 3, the user is not interested in the B and E values, so rules r_7 and r_{12} in Figure 4 can be dropped. Rules r_9 and r_{10} can also be removed, since the predicates in their heads are not used by other rules. Figure 10 shows the optimized program that can compute the same answer as before.

$$
\begin{array}{llll}
r_1: & ans(D) & \text{:-} & \widehat{v_1}(a_0, C), \widehat{v_3}(C, D) \\
r_2: & ans(D) & \text{:-} & \widehat{v_2}(a_0, B, C), \widehat{v_3}(C, D) \\
r_3: & \widehat{v_1}(A, C) & \text{:-} & domA(A), v_1(A, C) \\
r_4: & domC(C) & \text{:-} & domA(A), v_1(A, C) \\
r_5: & \widehat{v_2}(A, B, C) & \text{:-} & domC(C), v_2(A, B, C) \\
r_6: & domA(A) & \text{:-} & domC(C), v_2(A, B, C) \\
r_8: & \widehat{v_3}(C, D) & \text{:-} & domC(C), v_3(C, D) \\
r_{11}: & domC(C) & \text{:-} & v_4(C, E) \\
r_{15}: & domA(a_0) & \text{:-} &
\end{array}
$$

Fig. 10. The optimized datalog program in Example 3.

In general, the useless rules in $\Pi(\mathcal{Q}, \mathcal{V}_r)$ can be found as follows. Scan through all the rules in the program $\Pi(\mathcal{Q}, \mathcal{V}_r)$, except for the connection rules. For each rule r, check whether the IDB predicate in its head is used by other rules in the program. If not, rule r is useless and can be removed from the program. Repeat this process until no useless rules can be found in the program.

6. TESTING CONTAINMENT BETWEEN CONNECTIONS

We have shown so far how to trim useless source accesses for individual connections in a query. In this section, we study the containment problem between two connections in five steps:

(1) We formally define connection containment (Section 6.1).

(2) We prove that connection containment is decidable (Section 6.2).

(3) We introduce the question of *boundedness* for the program of a connection (Section 6.3). When one of the two programs in the containment test is bounded, we can perform the test efficiently.

(4) We develop a polynomial-time algorithm for testing connection boundedness (Section 6.4).

(5) Finally, we compare our decidability proof to the approach of [Millstein et al. 2000] (Section 6.5).

6.1 Connection containment

DEFINITION 7. *Let T be a connection in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$. The* program for connection T *is the program $\Pi(\mathcal{Q}_T, \mathcal{V})$, where $\mathcal{Q}_T$ is the query that has only one connection T, with the input attributes $I(\mathcal{Q})$ and the output attributes $O(\mathcal{Q})$. For any database $\mathcal{D}$ of the source relations, the ans facts computed by the program $\Pi(\mathcal{Q}_T, \mathcal{V})$, denoted $ANS(T, \mathcal{D})$, is the maximal answer to connection T.*

EXAMPLE 8. *Assume we have four sources of movie information as shown in Figure 11. Suppose that a user wants to find the addresses and dates of birth of the stars in movies produced by Disney. The query can be represented as*

$$\mathcal{Q} = \langle \{Studio\}, \{Addr, Dob\}, \{T_1, T_2\} \rangle$$

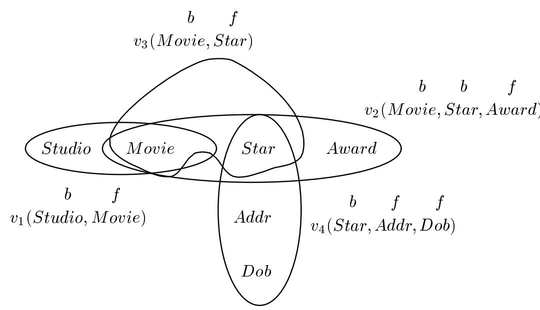

Fig. 11. The hypergraph representation of four movie sources.

r_1:	$ans(A, D)$	:- $\widehat{v_1}(disney, M), \widehat{v_2}(M, S, W), \widehat{v_4}(S, A, D)$
r_2:	$ans(A, D)$	:- $\widehat{v_1}(disney, M), \widehat{v_3}(M, S), \widehat{v_4}(S, A, D)$
r_3:	$\widehat{v_1}(T, M)$	:- $studio(T), v_1(T, M)$
r_4:	$movie(M)$	:- $studio(T), v_1(T, M)$
r_5:	$\widehat{v_2}(M, S, W)$	:- $movie(M), star(S), v_2(M, S, W)$
r_6:	$award(W)$	:- $movie(M), star(S), v_2(M, S, W)$
r_7:	$\widehat{v_3}(M, S)$	:- $movie(M), v_3(M, S)$
r_8:	$star(S)$	:- $movie(M), v_3(M, S)$
r_9:	$\widehat{v_4}(S, A, D)$	:- $star(S), v_4(S, A, D)$
r_{10}:	$addr(A)$	:- $star(S), v_4(S, A, D)$
r_{11}:	$dob(D)$	:- $star(S), v_4(S, A, D)$
r_{12}:	$studio(disney)$	:-

Fig. 12. The datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ for the query in Example 8.

in which the two connections are $T_1 = \{v_1, v_2, v_4\}$ *and* $T_2 = \{v_1, v_3, v_4\}$*. Clearly connection* T_2 *is independent, while* T_1 *is not. Figure 12 shows the corresponding datalog program* $\Pi(\mathcal{Q}, \mathcal{V})$*.*

Figure 13 shows the program $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ *for connection* T_1*. It can be constructed from the program* $\Pi(\mathcal{Q}, \mathcal{V})$ *by removing the connection rule* r_2*. That is, it includes the connection rule for* T_1*, and the* α*-rules, and the fact rule in* $\Pi(\mathcal{Q}, \mathcal{V})$*. Similarly,* $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ *can be constructed by removing the connection rule* r_1 *from the program* $\Pi(\mathcal{Q}, \mathcal{V})$ *in Figure 12.*

Although T_1 *and* T_2 *have different views, surprisingly, as we will see in Section 6.2, the answer to connection* T_1 *is* contained *in the answer to connection* T_2*, i.e.,* $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$*. Therefore, we can compute the answer to the query by considering only connection* T_2*, and thus save the queries to source* S_2*.*

Definition 8. *Suppose* T_1 *and* T_2 *are two connections in a query* $\mathcal{Q}$ *on source descriptions* $\mathcal{V}$*.* T_1 *is contained in* T_2*, denoted* $T_1 \subseteq^{ans} T_2$*, if the program* $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ *is contained in* $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ *with respect to the goal predicate ans; i.e., for any database* $\mathcal{D}$ *of* $\mathcal{V}$*, we have* $ANS(T_1, \mathcal{D}) \subseteq ANS(T_2, \mathcal{D})$*.*

This connection-containment problem is also called *relative containment* in [Millstein et al. 2000]. For brevity, further on, we use "connection containment" to mean "relative containment." In general, given two connections T_1 and T_2 in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$, we want to test whether $T_1 \subseteq^{ans} T_2$.

$$\begin{array}{lll}
r_1: & ans(A,D) & \text{:- } \widehat{v_1}(disney,M), \widehat{v_2}(M,S,W), \widehat{v_4}(S,A,D) \\
r_3: & \widehat{v_1}(T,M) & \text{:- } studio(T), v_1(T,M) \\
r_4: & movie(M) & \text{:- } studio(T), v_1(T,M) \\
r_5: & \widehat{v_2}(M,S,W) & \text{:- } movie(M), star(S), v_2(M,S,W) \\
r_6: & award(W) & \text{:- } movie(M), star(S), v_2(M,S,W) \\
r_7: & \widehat{v_3}(M,S) & \text{:- } movie(M), v_3(M,S) \\
r_8: & star(S) & \text{:- } movie(M), v_3(M,S) \\
r_9: & \widehat{v_4}(S,A,D) & \text{:- } star(S), v_4(S,A,D) \\
r_{10}: & addr(A) & \text{:- } star(S), v_4(S,A,D) \\
r_{11}: & dob(D) & \text{:- } star(S), v_4(S,A,D) \\
r_{12}: & studio(disney) & \text{:- }
\end{array}$$

Fig. 13. The program $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ for the connection T_1 in Example 8.

6.2 Connection containment is decidable

The program $\Pi(\mathcal{Q}_T, \mathcal{V})$ for a connection T could be recursive (as shown in Example 2), connection containment appears undecidable, since containment of datalog programs is undecidable [Shmueli 1993]. However, we prove connection containment *is* decidable, since it can be reduced to containment of monadic programs. A datalog program is *monadic* if its recursive IDB predicates are monadic (the nonrecursive predicates can have arbitrary arity). An IDB predicate is nonrecursive if it either does not depend on another IDB predicate, or it depends *only* on nonrecursive predicates. Under this definition of nonrecursive IDB predicates, it is shown in [Cosmadakis et al. 1988] that containment of monadic programs is decidable.

THEOREM 4. *Connection containment is decidable.*

The main idea of the proof is as follows. Let $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ and $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ be the programs for connection T_1 and T_2, respectively. We construct two programs Φ_1 and Φ_2, such that:

(1) Both Φ_1 and Φ_2 are monadic programs;
(2) $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ if and only if $\Phi_1 \subseteq \Phi_2$.

Since containment of monadic programs is decidable, the problem of testing $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ is decidable. The construction of Φ_1 from $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ has two steps. (Φ_2 can be constructed from $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ similarly.) In step (1), each α-predicate $\widehat{v_i}$ in the connection rule is substituted by the body of the corresponding α-rule of v_i, with the necessary variable unification. After the substitutions, remove the α-rules from $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$, and the new program, denoted $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})'$, is equivalent to $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$. Let r_0 be the modified connection rule. For instance, the program in Figure 13 can be rewritten to the following equivalent program:[2]

$$\begin{array}{lll}
r_0: & ans(A,D) & \text{:- } studio(disney), v_1(disney,M), movie(M), star(S), \\
 & & \quad v_2(M,S,W), v_4(S,A,D) \\
r_4: & movie(M) & \text{:- } studio(T), v_1(T,M) \\
r_6: & award(W) & \text{:- } movie(M), star(S), v_2(M,S,W) \\
r_8: & star(S) & \text{:- } movie(M), v_3(M,S)
\end{array}$$

[2] We use this example to illustrate how to construct monadic program Φ_1 for program $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$, even though the program $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ is this example is not recursive. In general, this program can be recursive, as shown in Example 2.

$$
\begin{array}{ll}
r_{10}\colon addr(A) & \text{:- } star(S), v_4(S,A,D) \\
r_{11}\colon dob(D) & \text{:- } star(S), v_4(S,A,D) \\
r_{12}\colon studio(disney) & \text{:-}
\end{array}
$$

In the new program, *ans* is the only IDB predicate that might not be unary. It could be recursive, since it might depend on recursive domain predicates. Thus this program is not monadic, and we cannot use the decidability result of monadic programs directly. Suppose the modified connection rule r_0 is:

$$ans(X_1, \ldots, X_m) \text{ :- } dom_1(Y_1), \ldots, dom_k(Y_k), F$$

where $dom_1, \ldots, dom_k$ are unary domain predicates, and F is a set of EDB formulas. In step (2), we construct program Φ_1 by replacing r_0 with the following rule:

$$final(Z) \text{ :- } E_1(X_1, Z), \ldots, E_m(X_m, Z), dom_1(Y_1), \ldots, dom_k(Y_k), F$$

where *final* is a new IDB predicate representing the answer to the new program, Z is a fresh variable, and $E_1, \ldots, E_m$ are new EDB predicates. For instance, the rule r_0 in the program above is replaced with:

$$
\begin{array}{ll}
r_0 : final(Z) \text{ :- } & E_1(A,Z), E_2(D,Z), studio(disney), v_1(disney, M), movie(M), \\
& star(S), v_2(M,S,W), v_4(S,A,D)
\end{array}
$$

Clearly the program Φ_1 is monadic, since all its IDB predicates are unary. Now we prove that $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ if and only if $\Phi_1 \subseteq \Phi_2$. The "only if" part is obvious by the construction of the two programs. For the "if" part, suppose $\Phi_1 \subseteq \Phi_2$, but $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \not\subseteq \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$. Then there exists a database D, such that there is a tuple $ans(x_1, \ldots, x_m)$ in $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})(D)$, but not in $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})(D)$. Let z be a fresh constant. We add tuples $E_1(x_1, z), \ldots, E_m(x_m, z)$ to D, and get a new database D'. By the construction of the two programs and D', tuple z is in $\Phi_1(D')$, but not in $\Phi_2(D')$, contradicting the fact that $\Phi_1 \subseteq \Phi_2$.

In conclusion, we can test $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ by testing $\Phi_1 \subseteq \Phi_2$, which is decidable since both Φ_1 and Φ_2 are monadic programs.

6.3 Connection boundedness

If one of the two programs in testing $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ is *bounded*, the containment can be tested efficiently using the algorithms in [Chandra and Merlin 1977; Chaudhuri and Vardi 1992; Sagiv and Yannakakis 1980]. A datalog program is *bounded* if it is equivalent to a finite union of conjunctive queries. For instance, the programs for the two queries in Example 8 are both bounded, because each of them can be rewritten to an equivalent conjunctive query. In this section, we study the following problem: given a connection T on source views $\mathcal{V}$ with binding restrictions, how do we test the boundedness of $\Pi(\mathcal{Q}_T, \mathcal{V})$? We develop a polynomial-time algorithm for testing boundedness of connections. The following example shows an unbounded connection.

EXAMPLE 9. *Consider the five source views in Figure 14. Assume a user knows the value of A is a, and wants to get the C values by joining the views v_1 and v_2. The following is the query:*

$$\mathcal{Q} = \langle \{A\}, \{C\}, \{T\} \rangle$$

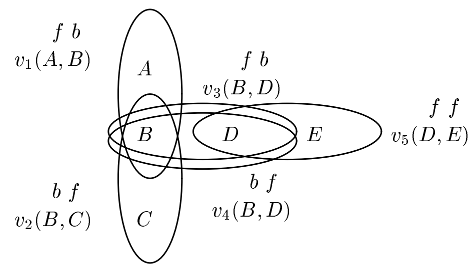

Fig. 14. The source descriptions in Example 9.

in which the only connection is $T = \{v_1, v_2\}$. Assume the five attributes have five different domains. The program $\Pi(\mathcal{Q}_T, \mathcal{V})$ is shown in Figure 15. This program is unbounded. Intuitively, since the binding pattern of $v_3(B, D)$ is fb, and the binding pattern of $v_4(B, D)$ is bf, we can visit these two source views repeatedly to retrieve more B values. Each new B value can participate in $v_1 \bowtie v_2$ and generate more answers to the query. (We will give a proof of the unboundedness in Section 6.4.)

r_1:	$ans(C)$	:- $\widehat{v_1}(a, B), \widehat{v_2}(B, C)$	r_8:	$\widehat{v_4}(B, D)$	:- $domB(B), v_4(B, D)$
r_2:	$\widehat{v_1}(A, B)$	:- $domB(B), v_1(A, B)$	r_9:	$domD(D)$	:- $domB(B), v_4(B, D)$
r_3:	$domA(A)$	:- $domB(B), v_1(A, B)$	r_{10}:	$\widehat{v_5}(D, E)$	:- $v_5(D, E)$
r_4:	$\widehat{v_2}(B, C)$	:- $domB(B), v_2(B, C)$	r_{11}:	$domD(D)$	:- $v_5(D, E)$
r_5:	$domC(C)$	:- $domB(B), v_2(B, C)$	r_{12}:	$domE(E)$	:- $v_5(D, E)$
r_6:	$\widehat{v_3}(B, D)$	:- $domD(D), v_3(B, D)$	r_{13}:	$domA(a)$	:-
r_7:	$domB(B)$	:- $domD(D), v_3(B, D)$			

Fig. 15. The program $\Pi(\mathcal{Q}_T, \mathcal{V})$ in Example 9.

6.4 Testing connection boundedness

In this section, we develop a polynomial-time algorithm for testing connection boundedness, even though boundedness of datalog programs in general is undecidable [Gaifman et al. 1993].

6.4.1 *Independent connections.* Recall that a connection T in a query $\mathcal{Q}$ is *independent* if *f-closure*$(I(\mathcal{Q}), T) = T$. For instance, the connection T_2 in Example 8 is independent, while connection T_1 is not. Similarly, the connection in Example 9 is not independent.

THEOREM 5. *If a connection T is independent, then T is bounded.*

PROOF. Assume $T = \{w_1, \ldots, w_k\}$. Since *f-closure*$(I(\mathcal{Q}), T) = T$, there exists a sequence of the views in connection T, say, $w_{i_1}, \cdots, w_{i_k}$, that satisfies: (i) $\mathcal{B}(w_{i_1}) \subseteq I(\mathcal{Q})$; (ii) for $j = 2, \ldots, k$, $\mathcal{B}(w_{i_j}) \subseteq I(\mathcal{Q}) \cup \mathcal{A}(w_{i_1}) \cup \cdots \cup \mathcal{A}(w_{i_{j-1}})$. For any database of $\mathcal{V}$, we can compute the maximal answer to T as follows. Compute the corresponding sequence of n *supplementary relations* [Beeri and Ramakrishnan 1987] $I_1, \ldots, I_n$, where I_i is the supplementary relation after the first i subgoals have been processed. The supplementary relation I_n is the answer to query Q. Therefore, we can compute the answer to T after $n + 1$ applications of the rules in $\Pi(\mathcal{Q}_T, \mathcal{V})$ (the last application is to evaluate the connection rule). ☐

If a connection T is not independent, the predicate *ans* in $\Pi(\mathcal{Q}_T, \mathcal{V})$ may not be bounded, as shown in Example 9.

6.4.2 *BF-loop and BF-graph*

Definition 9. *A sequence of views forms a* BF-loop *if it forms a BF-chain, and the bound attributes of the head overlap with the free attributes of the tail.*

For instance, in Figure 14, (v_3, v_4) forms a BF-loop, because $\mathcal{F}(v_3) \cap \mathcal{B}(v_4) = \{B\}$ and $\mathcal{F}(v_4) \cap \mathcal{B}(v_3) = \{D\}$.

Definition 10. *The BF-graph of a set of source views $\mathcal{W}$ is a directed graph in which each vertex corresponds to a view in $\mathcal{W}$, and there is an edge from vertex v_i to vertex v_j if and only if $\mathcal{F}(v_i) \cap \mathcal{B}(v_j) \neq \emptyset$.*

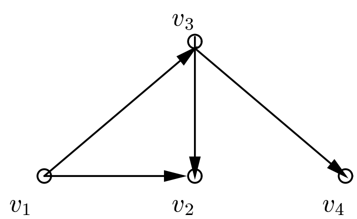

Fig. 16. The BF-graph for Example 8.

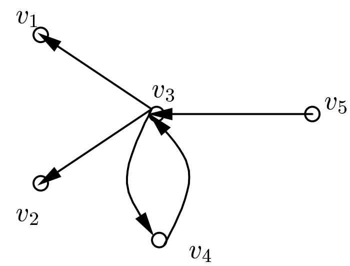

Fig. 17. The BF-graph for Example 9.

Figures 16 and 17 show the BF-graphs of the source views in Example 8 and Example 9, respectively. For instance, in Figure 16, there is an edge from vertex v_1 to vertex v_2 because $\mathcal{F}(v_1) \cap \mathcal{B}(v_2) = \{Movie\}$. Clearly there is a BF-loop in a set of source views if and only if the BF-graph of these views is cyclic.

6.4.3 *The algorithm TestBoundedness*

Theorem 6. *If T is a connection in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$, and all the source views on T are queryable, then T is bounded if and only if there is no BF-loop among the views in b-closure($\mathcal{K}$), in which $\mathcal{K}$ is a kernel of T.*

Proof. *If*: Assume $T = \{w_1, \ldots, w_n\}$, and *b-closure*($\mathcal{K}$) $= \{v_1, \ldots, v_k\}$. Since there is no BF-loop among the views in *b-closure*($\mathcal{K}$), there exists a BF-chain $v_{i_1}, \ldots, v_{i_k}$ in *b-closure*($\mathcal{K}$) with distinct views, such that the free attributes of each view v_{i_j} do not overlap with the bound attributes of any previous source view. Starting with the initial bindings in $\mathcal{Q}$ and following the sequence $v_{i_1}, \ldots, v_{i_k}$, we use the views in this sequence to send source queries and retrieve all the possible bindings X of the attributes in $\mathcal{K}$. With these bindings X and the initial bindings in $I(\mathcal{Q})$, there exists a sequence of the views in T, say $w_{l_1}, \ldots, w_{l_n}$, such that the binding requirements of each view in the sequence can be satisfied. We follow this sequence to send source queries, collect tuples from the sources in the connection, and evaluate the connection rule in $\Pi(\mathcal{Q}_T, \mathcal{V})$ to compute the answer to T. Therefore, we can evaluate the rules in a finite number of steps to compute the answer to the connection, and the number is independent of the source relations. Thus T is bounded.

Only If: If there is a BF-loop among the views *b-closure*($\mathcal{K}$), we prove T is unbounded by showing that for any integer $k > 0$, there exists some database $\mathcal{D}$, such that only after k applications of the rules in $\Pi(\mathcal{Q}_T, \mathcal{V})$ we can compute a tuple in $ANS(T, \mathcal{D})$. Since there is a BF-loop among *b-closure*($\mathcal{K}$), there exists an attribute A in $\mathcal{K}$, such that there is a BF-loop among *b-closure*(A). For any integer $k > 0$, there is a BF-chain $v_1, \ldots, v_k$ with length k, and $A \in \mathcal{F}(v_k)$. We can add tuples to the relations on the BF-chain, such that in following the BF-chain only, we can retrieve a tuple in $ANS(T, \mathcal{D})$. In other words, we "populate" the relations in a BF-loop of the views in *b-closure*($\mathcal{K}$) along the loop as many times as we want. By the construction of database $\mathcal{D}$, we can only compute a tuple in $ANS(T, \mathcal{D})$ after k applications of the rules in $\Pi(\mathcal{Q}_T, \mathcal{V})$. □

Consider the two connections in Example 8. Connection T_1 has one kernel $\{Star\}$, whose backward-closure is $\{v_1, v_3\}$. Clearly there is no BF-loop in $\{v_1, v_3\}$, thus T_1 is bounded. Similarly, connection T_2 has one kernel $\emptyset$, and there is no BF-loop in its backward-closure, thus T_2 is also bounded. Thus the two programs $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ and $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ can be rewritten to unbounded queries. In Example 9, $\{B\}$ is the only kernel of the connection $\{v_1, v_2\}$, and the backward-closure of $\{B\}$ is $\{v_1, v_3, v_4, v_5\}$. Since there is a BF-loop, (v_3, v_4), among these four views, by Theorem 6, the connection is unbounded. If a connection T is independent, it has only one kernel, the empty set $\emptyset$. Thus the backward-closure of this kernel is empty, and there is no BF-loop among the views in the backward-closure. By Theorem 6, connection T is bounded, which is consistent with Theorem 5.

By Theorem 6, we give an algorithm called *TestBoundedness* for testing connection boundedness, as shown in Figure 18.

Algorithm TestBoundedness: Test connection boundedness
Input: • $\mathcal{V}$: Source views with binding restrictions.
• $\mathcal{Q}$: A query on $\mathcal{V}$.
• T: A connection in $\mathcal{Q}$.
Output: Decision about the boundedness of T.
Method:
(1) Compute the queryable views $\mathcal{V}_q = f\text{-}closure(I(\mathcal{Q}), \mathcal{V})$.
(2) If there is one view $v \in T$ that is not in $\mathcal{V}_q$, T is bounded and return.
(3) Compute a kernel $\mathcal{K}$ of T.
(4) Compute *b-closure*($\mathcal{K}$).
(5) Build the BF-graph G of *b-closure*($\mathcal{K}$).
(6) Test the acyclicity of G. If G is acyclic, then T is bounded; otherwise, T is unbounded.

Fig. 18. Testing the boundedness of a connection.

Let us see the complexity of this algorithm. Assume $\mathcal{V}$ has n views, T has m views and k attributes, and *b-closure*($\mathcal{K}$) has p views. Section 5 gives the details how steps 1 to 4 are executed in $O(kn^2)$ time. Steps 5 and 6 can be done in $O(p^2)$ time, since we can test the cyclicity of the BF-graph in $O(p^2)$ time [Aho et al. 1983]. Therefore, the complexity of the algorithm TestBoundedness is:

$$O(kn^2) + O(p^2) = O(kn^2)$$

6.5 Comparison

The reader should compare our decidability proof of Theorem 4 to the approach in [Millstein et al. 2000]. Besides the fact that we studied this problem in 1999, the following are the differences between these two approaches:

(1) [Millstein et al. 2000] uses the source-centric approach to information integration, while we use the query-centric approach. However, our proof can be generalized to the source-centric approach [Li and Chang 1999].
(2) The decidability proof in [Millstein et al. 2000] is based on the assumption that the set of bindings for the contained query is a subset of the bindings for the containing query. Theorem 4 is true even if the two queries have different initial bindings. However, we assume both queries are connection queries, while in [Millstein et al. 2000] the contained query can be a recursive datalog program.
(3) We also discuss how to test the boundedness of the program for a query.

7. CONCLUSION AND DISCUSSION

In information-integration systems, especially in the World Wide Web, sources may have restrictions on retrieving their information. In this paper we showed that sources not mentioned in a query can contribute to the query's result by providing useful bindings. We proposed a query-planning framework in the presence of source restrictions. In the framework, a user query and source descriptions are translated into a datalog program, which can be evaluated on the source relations to answer the query. We then solved optimization problems in this framework. In particular, we showed in what cases accessing off-query sources is necessary, and developed an algorithm for finding all useful sources for a query. We also solved the problem of testing whether the answer to a query is contained in the answer to another query. By using the results on monadic programs, we proved this containment problem is decidable. We developed an efficient algorithm for testing program boundedness in our framework. We can perform the containment test efficiently when one of the programs in the test is bounded.

Other Possibilities For Obtaining bindings

Theorem 1 suggests that accessing off-connection views is only necessary for non-independent connections. So far, we have assumed that the bindings of a domain are either from a user query or from other source queries. If cached data is available, it can be incorporated into the program $\Pi(\mathcal{Q}, \mathcal{V})$ for a query $\mathcal{Q}$ and source descriptions $\mathcal{V}$. Suppose that we have a cached tuple $t_i(a_1, \ldots, a_n)$ for source view $v_i(A_1, \ldots, A_n)$. The following rules are added to the program $\Pi(\mathcal{Q}, \mathcal{V})$:

$$\begin{array}{lll} \widehat{v_1}(a_1, \ldots, a_n) & :- & \\ domA_i(a_i) & :- & \quad (i = 1, \ldots, n) \end{array}$$

The predicates $domA_1, \ldots, domA_n$ are the domain predicates for the attributes $A_1, \ldots, A_n$, respectively. The first rule says that tuple $t_i(a_1, \ldots, a_n)$ is an obtained tuple of source view v_i. The other fact rules represent the bindings for the corresponding domains. The new rules can contribute more answers to the query. Some views that were nonqueryable when we considered only the initial bindings in $\mathcal{Q}$

may now become queryable with the new bindings from the cached data. In general, if we have some information about a domain, we can always incorporate the information into the program $\Pi(\mathcal{Q}, \mathcal{V})$ by adding the corresponding fact rules.

We may also obtain bindings by using some known domain knowledge. For example, suppose that we have a source view $student(name, dept, GPA)$ with the binding pattern *bbf*. That is, every query to this source must supply a name and a department of a student, so that the student's GPA can be returned. Assume we know that all the students at the source are in four departments: $\{$*CS, EE, Physics, Chemistry*$\}$. Then we can use these four departments as bindings for attribute *dept* to query the source, and we do not need other sources to retrieve *dept* bindings.

Computing a partial answer

In some cases a user may be interested in a partial answer to a query. Thus we do not need to compute the maximal answer, which may be expensive to retrieve. Theorem 1 suggests that if a connection is independent, its complete answer can be computed by using only the views in the query. If a connection T is not independent, we can find a kernel $\mathcal{K}$ of T. We access some sources in *b-closure*$(\mathcal{K})$ to obtain bindings for the attributes in $\mathcal{K}$, and compute a partial answer for the connection. Notice that we may access only a subset of the backward-closure of $\mathcal{K}$, since we are not interested in the maximal answer for T. In addition, we need to consider the tradeoff between the number of results and the number of source accesses. The more sources we access, the more bindings we can retrieve, and the more answers we can compute for the connection. We decide how many source queries to send based on how many results the user is interested in.

Extending results to conjunctive queries

Some results on connection queries in this paper can be extended to arbitrary conjunctive queries. (1) We can construct a datalog program for a conjunctive query following the idea in Section 3.1. That is, we introduce an IDB predicate for each domain that can be shared by several attributes. The idea of using α-predicates and adding rules can be generalized naturally. (2) The decidability result in Section 6 can be extended to datalog programs for conjunctive queries.

Open problems

Currently we are working on the following problems. (1) For the datalog program of a conjunctive query, it is still not clear how to decide which relations are relevant to the program. The "kernel" concept in Section 5 might give us a hint on how to trim useless source relations. (2) We want to know how to test the boundedness of the program of a conjunctive query. (3) In addition, it is an open problem how to evaluate the program of a query efficiently. We might use existing algorithms for evaluating datalog programs [Bancilhon and Ramakrishnan 1986], such as the magic-sets techniques [Beeri and Ramakrishnan 1987]. Finding efficient algorithms for evaluating the program of a query in our framework deserves investigation.

ACKNOWLEDGMENTS

We thank Jeff Ullman for his valuable comments on this material. We thank Moshe Vardi for clarifying the definition of monadic programs in [Cosmadakis et al. 1988].

We thank Foto Afrati for helping us prove Theorem 4. We are very grateful to the anonymous referees for their helpful comments.

REFERENCES

ABITEBOUL, S. AND DUSCHKA, O. M. 1998. Complexity of answering queries using materialized views. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)* (1998), pp. 254–263.

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. 1983. *Data Structures and Algorithms.* Addison-Wesley Publishing Company.

BANCILHON, F. AND RAMAKRISHNAN, R. 1986. An amateur's introduction to recursive query processing strategies. In *Proc. of ACM SIGMOD* (1986), pp. 16–52.

BAYARDO JR., R. J. ET AL. 1997. Infosleuth: Semantic integration of information in open and dynamic environments (experience paper). In *Proc. of ACM SIGMOD* (1997), pp. 195–206.

BEERI, C. AND RAMAKRISHNAN, R. 1987. On the power of magic. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)* (1987), pp. 269–283.

CAREY, M. J., HAAS, L. M., SCHWARZ, P. M., ARYA, M., CODY, W. F., FAGIN, R., FLICKNER, M., LUNIEWSKI, A., NIBLACK, W., PETKOVIC, D., II, J. T., WILLIAMS, J. H., AND WIMMERS, E. L. 1995. Towards heterogeneous multimedia information systems: The garlic approach. In *RIDE-DOM* (1995), pp. 124–131.

CHANDRA, A. K. AND MERLIN, P. M. 1977. Optimal implementations of conjunctive queries in relational data bases. In *Proceedings of the 9th ACM Symposium on Theory of Computing (STOC)* (1977), pp. 77–90. ACM Press.

CHAUDHURI, S. AND VARDI, M. Y. 1992. On the equivalence of recursive and nonrecursive datalog programs. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)* (1992), pp. 55–66.

CHAWATHE, S., GARCIA-MOLINA, H., HAMMER, J., IRELAND, K., PAPAKONSTANTINOU, Y., ULLMAN, J. D., AND WIDOM, J. 1994. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan* (Tokyo, Japan, 1994), pp. 7–18.

CLUET, S., DELOBEL, C., SIMEON, J., AND SMAGA, K. 1998. Your mediators need data conversion! In *Proc. of ACM SIGMOD* (1998), pp. 177–188.

COSMADAKIS, S. S., GAIFMAN, H., KANELLAKIS, P. C., AND VARDI, M. Y. 1988. Decidable optimization problems for database logic programs. In *Proc. 20th ACM Symp. on Theory of Computing* (1988), pp. 477–490.

DUSCHKA, O. M. 1998. Query Planning and Optimization in Information Integration. Ph.D thesis, Stanford University, Stanford, California.

DUSCHKA, O. M. AND GENESERETH, M. R. 1997. Answering recursive queries using views. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)* (1997), pp. 109–116. ACM Press.

DUSCHKA, O. M. AND LEVY, A. Y. 1997. Recursive plans for information gathering. In *15th International Joint Conference on Artificial Intelligence* (Nagoya, Japan, 1997), pp. 778–784.

FLORESCU, D., LEVY, A., MANOLESCU, I., AND SUCIU, D. 1999. Query optimization in the presence of limited access patterns. In *Proc. of ACM SIGMOD* (1999), pp. 311–322.

GAIFMAN, H., MAIRSON, H., SAGIV, Y., AND VARDI, M. Y. 1993. Undecidable optimization problems for database logic programs. *Journal of the ACM 40*, 3, 683–713.

GENESERETH, M. R., KELLER, A. M., AND DUSCHKA, O. M. 1997. Infomaster: An information integration system. In *Proc. of ACM SIGMOD* (1997), pp. 539–542.

HAAS, L. M., KOSSMANN, D., WIMMERS, E. L., AND YANG, J. 1997. Optimizing queries across diverse data sources. In *Proc. of VLDB* (1997), pp. 276–285.

HAMMER, J., GARCÍA-MOLINA, H., NESTOROV, S., YERNENI, R., BREUNIG, M., AND VASSALOS, V. 1997. Template-based wrappers in the TSIMMIS system. In *Proc. of ACM SIGMOD* (1997), pp. 532–535.

Ives, Z., Florescu, D., Friedman, M., Levy, A., and Weld, D. 1999. An adaptive query execution engine for data integration. In *Proc. of ACM SIGMOD* (1999), pp. 299–310.

Levy, A. Y., Mendelzon, A. O., Sagiv, Y., and Srivastava, D. 1995. Answering queries using views. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)* (1995), pp. 95–104.

Levy, A. Y., Rajaraman, A., and Ordille, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB* (1996), pp. 251–262.

Li, C. and Chang, E. 1999. Testing query containment in the presence of limited access patterns. In *Technical report, Computer Science Dept., Stanford Univ.* (1999).

Li, C. and Chang, E. 2000. Query planning with limited source capabilities. In *International Conference on Data Engineering (ICDE)* (2000), pp. 401–412.

Li, C. and Chang, E. 2001. On answering queries in the presence of limited access patterns. In *International Conference on Database Theory (ICDT)* (2001), pp. 99–113.

Li, C., Yerneni, R., Vassalos, V., Garcia-Molina, H., Papakonstantinou, Y., Ullman, J. D., and Valiveti, M. 1998. Capability based mediation in TSIMMIS. In *Proc. of ACM SIGMOD* (1998), pp. 564–566.

Maluf, D. A. and Wiederhold, G. 1997. Abstraction of representation for interoperation. In *International Symposium on Methodologies for Intelligent Systems (ISMIS)* (1997), pp. 441–455.

Millstein, T., Levy, A., and Friedman, M. 2000. Query containment for data integration systems. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)* (2000).

Papakonstantinou, Y., Garcia-Molina, H., and Widom, J. 1995. Object exchange across heterogeneous information sources. In P. S. Yu and A. L. P. Chen Eds., *11th Conference on Data Engineering* (Taipei, Taiwan, 1995), pp. 251–260. IEEE Computer Society.

Qian, X. 1996. Query folding. In *International Conference on Data Engineering (ICDE)* (1996), pp. 48–55.

Rajaraman, A., Sagiv, Y., and Ullman, J. D. 1995. Answering queries using templates with binding patterns. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)* (1995), pp. 105–112.

Sagiv, Y. and Yannakakis, M. 1980. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM 27*, 4, 633–655.

Shmueli, O. 1993. Equivalence of datalog queries is undecidable. *Journal of Logic Programming 15*, 3, 231–241.

Tomasic, A., Raschid, L., and Valduriez, P. 1998. Scaling access to heterogeneous data sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering 10*, 5, 808–823.

Ullman, J. D. 1989. *Principles of Database and Knowledge-base Systems, Volumes II: The New Technologies.* Computer Science Press, New York.

Ullman, J. D. 1997. Information integration using logical views. In *International Conference on Database Theory (ICDT)* (1997), pp. 19–40.

Vassalos, V. and Papakonstantinou, Y. 1997. Describing and using query capabilities of heterogeneous sources. In *Proc. of VLDB* (1997), pp. 256–265.

Wiederhold, G. 1992. Mediators in the architecture of future information systems. *IEEE Computer 25*, 3, 38–49.

Yerneni, R., Li, C., Garcia-Molina, H., and Ullman, J. D. 1999. Computing capabilities of mediators. In *Proc. of ACM SIGMOD* (1999), pp. 443–454.

Yerneni, R., Li, C., Ullman, J. D., and Garcia-Molina, H. 1999. Optimizing large join queries in mediation systems. In *International Conference on Database Theory (ICDT)* (1999), pp. 348–364.