Testing Query Containment in the Presence of Binding Restrictions (Technical Report)

Chen Li
Department of Computer Science
Stanford University
chenli@db.stanford.edu

December 22, 1999

Abstract

In information-integration systems, sources have diverse and limited query capabilities. In a recent paper [LC00], we showed that sources not mentioned in a query can contribute to the query result by providing useful bindings. We studied *connection queries*, where each connection query is a natural join of distinct source views with the necessary selection and projection. Some optimization problems are left open, including whether the answer computed by one connection is contained in that computed by another connection. In this paper we study this connection-containment problem. Since [LC00] often produces a recursive Datalog program to answer a connection query optimally, containment seems undecidable. However, because the Datalog programs of [LC00] have a special form, their containment can be reduced to containment of monadic programs, which is known to be decidable. Further, the decidability of monadic Datalog programs involves a complex algorithm. We therefore introduce the question of boundedness for the programs of [LC00], and show that boundedness is decidable in polynomial time. We then show in addition that when the contained program is bounded, we have a much simpler algorithm for performing the containment test.

Keywords: information-integration systems, binding restrictions, query containment, query equivalence, Datalog programs.

1 Introduction

The goal of information-integration systems is to support seamless access to heterogeneous data sources. Many systems (e.g., [C+94, GKD97, HKWY97, IFF+99, LRO96, TRV98]) have been proposed to reach this goal, and information-integration technologies have been successfully used in many E-commerce comparison shops on the World Wide Web, such as `mySimon.com` [myS], and `cnet.com` [cne]. In heterogeneous environments, especially in the context of the World Wide Web, sources have diverse and limited query capabilities. For example, many Web bookstores like `amazon.com` [Ama] and `barnesandnoble.com` [Bar] provide only search forms. A user fills out a form by specifying the values of some attributes, e.g., book title, author name, publisher, or ISBN, so that the source returns the books satisfying the query conditions. These sources do not accept queries such as "return the information about all the books you know about."

In a recent paper [LC00], we showed that sources not mentioned in a query can contribute to the query result by providing useful bindings. We proposed a query-planning framework, in which a query and source views with binding restrictions are translated into a Datalog program [Ull89]. We can evaluate the program on the source relations to compute the maximal obtainable answer to the query. We showed in which cases accessing off-query sources is necessary, and where it is necessary, we developed an algorithm for finding the useful sources for a query.

In that paper we studied *connection queries*, where each connection query is a natural join of distinct source views with the necessary selection and projection. In this paper we address some optimization problems left open in that paper, including whether the answer computed by one connection is contained in that by another connection. The following is a motivating example.

EXAMPLE 1.1 Assume we have four sources as shown in Table 1. For instance, source S_1 has the information about studios and their movies, and source S_2 has the information about awards of movies and stars. Note that a source might store only partial information about its corresponding domain. For instance, there might be a star s in a movie m, but the pair (m, s) is not stored at the source S_3. The "Must Bind" column indicates the attributes that must be specified values at a source. For instance, every source query to S_1 must specify a studio name; every source query to S_2 must provide a movie title and a star name.

Table 1: Four sources.

Source	Contents	Must Bind
S_1	$\mathtt{v_1(Studio, Movie)}$	`Studio`
S_2	$\mathtt{v_2(Movie, Star, Award)}$	`Movie` and `Star`
S_3	$\mathtt{v_3(Movie, Star)}$	`Movie`
S_4	$\mathtt{v_4(Star, Addr)}$	`Star`

Suppose that a user wants to find the addresses of the stars in movies produced by Disney. The query is the union of the following two conjunctive queries (CQs for short):

Q_1: $\mathtt{ans(A)}$:- $\mathtt{v_1(disney, M), v_2(M, S, W), v_4(S, A)}$
Q_2: $\mathtt{ans(A)}$:- $\mathtt{v_1(disney, M), v_3(M, S), v_4(S, A)}$

The query joins the views along two *connections*: $T_1 = \{v_1, v_2, v_4\}$ and $T_2 = \{v_1, v_3, v_4\}$. Given the source restrictions, we can answer Q_2 using only the views in connection T_2 as follows. Send source S_1 a query $v_1(disney, M)$ to retrieve movies produced by Disney. For each returned movie m, send S_3 a query $v_3(m, S)$ to obtain the stars in movie m. For each returned star s, send source S_4 a query $v_4(s, A)$ to retrieve the address of star s. To answer Q_1, we cannot obtain any binding for the attribute *Star* if we use only the views in connection T_1. However, we can use the *Star* bindings produced from v_3 to compute some answer to Q_1, although v_3 itself is not in T_1.

Although connections T_1 and T_2 have different views, surprisingly, as we will see in Section 3, if all the available information about studios, movies, and stars is from the user query and the four sources, the answer that can be computed by connection T_1 is *contained* in the answer by connection T_2. Therefore, we can compute the answer to the query $Q_1 \cup Q_2$ by only answering the query Q_2, and we do not need to send any queries to source S_2. □

In this paper we study the following connection-containment problem: "Is the answer computed by one connection contained in the answer by another connection?" Being able to test connection containment can save us unnecessary source queries, as shown in Example 1.1. Although a connection query is a CQ, we cannot use existing techniques of containment between CQs (e.g., [CM77, SY80]) to solve our problem directly, because [LC00] often produces a recursive Datalog program to answer a connection query optimally. Furthermore, since it is known that containment of Datalog programs is undecidable [Shm93], our connection-containment problem appears undecidable. However, the Datalog programs of [LC00] have a special form. In particular, we show that given source views with binding restrictions, each connection query can be translated into an equivalent monadic program. A program is *monadic* if every IDB predicate in the program is monadic (with arity one); its nonrecursive IDB predicates can have arbitrary arity. Therefore, our containment problem can be reduced to containment of *monadic programs*, which is known to be decidable [CGKV88].

[CGKV88] involved a complex algorithm (using tree-automata theory) to solve the containment problem of monadic programs. We introduce the question of *boundedness* for the programs of [LC00]. A program is *bounded* if it is equivalent to a finite union of CQs [NS87]. It is known that boundedness of monadic Datalog programs is decidable [CGKV88], although this boundedness problem is undecidable in general [GMSV93]. We develop a polynomial-time algorithm to test the boundedness of a connection. In addition, we show that when the program of the contained connection is bounded, we have a much simpler algorithm for performing the containment test.

The rest of the paper is organized as follows. Section 2 gives the notation used throughout the paper, and reviews how to translate a query and source descriptions into a Datalog program that can be evaluated on the source relations. It also gives the formal definition of the connection-containment problem. In Section 3 we show that connection containment is decidable, and how to test connection containment when the contained connection is bounded. In Section 4 we propose an algorithm for testing the boundedness of a connection. We give the conclusions in Section 5.

1.1 Related Work

Many studies have been done on query planning in the presence of limited source capabilities. Some early studies [DL97, RSU95] are done in the context of answering queries using views. In the presence of binding restrictions, [LYV+98] showed how to find a feasible plan to answer a CQ; [YLUGM99, FLMS99] studied how to optimize CQs; [Li99] developed algorithms for testing whether the complete answer to a query can be computed; [DL97] showed how to add domain rules and inverse rules of source descriptions to a Datalog query to construct a new program, which can be evaluated on the sources. Our earlier paper [LC00] showed how to translate connection queries to Datalog programs while incorporating the source restrictions into the programs.[1] In this paper we solve some open optimization problems in [LC00], and our techniques can also be generalized to apply to the programs in [DL97].

A closely related study is containment of monadic programs [CGKV88]. We use its result to prove the decidability of our connection-containment problem. Other related studies include containment of CQs [CM77], containment between unions of CQs [SY80], and containment between a CQ and a Datalog program [Sag88]. In this study we use these results to solve our connection-containment problem when the connections meet some boundedness conditions.

[1]There are some differences between programs generated by the algorithms in [DL97] and [LC00]. For example, [DL97] introduced one domain predicate for all the attributes, while[LC00] used one domain predicate for each attribute by assuming different attributes are from different domains.

2 Preliminaries

In this section, we introduce the notation used throughout the paper. We show how to translate a query and source descriptions into a Datalog program to compute the maximal answer to the query. We also give the formal definition of connection containment.

2.1 Source Views and Queries

Let an integration system have n sources, say, $S_1, \ldots, S_n$. Suppose each source S_i uses relational model and provides its data in the form of a view v_i. If sources have other data models, we can use *wrappers* [HGMN+97] to create the simple relational view of data. In the case where a source has multiple relations, we can represent this source by multiple logical sources, each of which exports only one relation. The query capability of a source is described as a template with a binding pattern representing the possible queries that the source can accept [Ull89]. The adornments for the attributes in the binding pattern include b (the attribute must be bound) and f (the attribute can be free). In Example 1.1, the binding pattern for the view $v_1(Studio, Movie)$ is bf, and the binding pattern for the view $v_2(Movie, Star, Award)$ is bbf. We assume that each source relation has only one template, and we use v_i to stand for both the source view and its adorned template. Let $\mathcal{V}$ denote the source views with their adornments ("source descriptions" for short), and $\mathcal{D}$ be a database of $\mathcal{V}$. We assume that differences in ontologies, vocabularies, and formats used by sources have been resolved. In particular, if two sources share an attribute name, then the attributes are equivalent (i.e., wrappers take care of any differences). Related research [MW97, PGMW95] suggests ways to deal with ontology and format differences.

Source-view schemas can be represented by a hypergraph [Ull89], in which each node is an attribute and each hyperedge is a source view. For instance, the hypergraph of the four source views is shown in Figure 1, which also shows the adornments of the attributes in each source view.

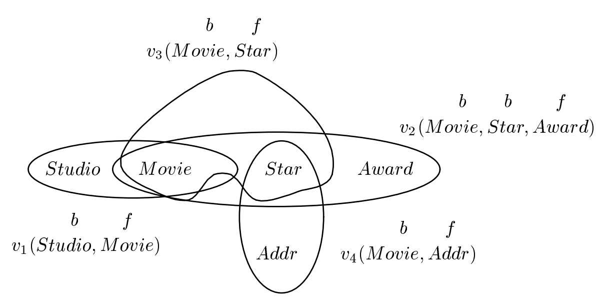

Figure 1: The hypergraph representation

A user query is represented in the form

$$\mathcal{Q} = <\mathcal{I}, \mathcal{O}, \mathcal{T}>$$

where $\mathcal{I}$ is a list of input assignments of the form `attribute = constant`, $\mathcal{O}$ is a list of output attributes whose values the user is interested in, and $\mathcal{T}$ is a list of *connections*. Each connection is a set of source views that connects the input attributes and the output attributes. For instance, the query in Example 1.1 can be represented as

$$\mathcal{Q} = <\{Studio\}, \{Addr\}, \{T_1, T_2\}>$$

in which the two connections are $T_1 = \{v_1, v_2, v_4\}$ and $T_2 = \{v_1, v_3, v_4\}$. For those tuples in the natural join of the relations in each connection T that satisfy the input constraints in $\mathcal{Q}$, their projections onto the output attributes are the *complete answer for connection* T.

Let $I(\mathcal{Q})$ and $O(\mathcal{Q})$ respectively denote the input attributes and the output attributes of a query $\mathcal{Q}$. Let $\mathcal{A}(\mathcal{Q}) = I(\mathcal{Q}) \cup O(\mathcal{Q})$ denote all the attributes of $\mathcal{Q}$, and $\mathcal{A}(T)$ be all the attributes of the views in a connection T in $\mathcal{Q}$. In Example 1.1, $I(\mathcal{Q}) = \{Studio\}$, $O(\mathcal{Q}) = \{Addr\}$, $\mathcal{A}(\mathcal{Q}) = \{Studio, Addr\}$, and $\mathcal{A}(T_1) = \{Studio, Movie, Star, Award, Addr\}$. Note that there can be multiple input attributes and multiple output attributes in a query.

2.2 Constructing the Program $\Pi(\mathcal{Q}, \mathcal{V})$

We showed in [LC00] how to translate source descriptions $\mathcal{V}$ and a query $\mathcal{Q}$ into a *Datalog program for* $\mathcal{Q}$, denoted $\Pi(\mathcal{Q}, \mathcal{V})$. The program $\Pi(\mathcal{Q}, \mathcal{V})$ incorporates the source restrictions and the query, and thus can be evaluated on the source relations to compute the maximal answer to the query. For instance, Figure 2 shows the Datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ for the four views and the query in Example 1.1.

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

Figure 2: The Datalog program $\Pi(\mathcal{Q}, \mathcal{V})$ for the query in Example 1.1

Program $\Pi(\mathcal{Q}, \mathcal{V})$ is constructed as follows. For each source view v_i, we introduce an EDB predicate ([Ull89]) 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 $T = \{v_1, \dots, v_k\}$ be a connection in $\mathcal{Q}$. The following rule is the *connection rule* of T:

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

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 and r_2 are the connection rules of the connections T_1 and T_2, 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 *studio*, *movie*, *star*, and *addr* represent the corresponding domains, 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 $k = p+1, \ldots, l$, 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_3 in Figure 2 is the α-rule of v_1; r_4 is its domain rule. 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_{11} in Figure 2 is a fact rule of attribute *studio*. 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 and r_2 are two connection rules. Rules r_3, r_5, r_7, and r_9 are the α-rules of the predicates v_1, v_2, v_3, and v_4, respectively. Rules r_4, r_6, r_8, and r_{10} are the corresponding domain-rules. Rule r_{11} is a fact rule for the input attribute *Studio*. During the construction of the program $\Pi(\mathcal{Q}, \mathcal{V})$, we make the following important binding assumptions:

1. Each binding for an attribute must be from its domain.
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.

If we have more information about an attribute, we can incorporate the new information into the program $\Pi(\mathcal{Q}, \mathcal{V})$ by introducing the corresponding fact rules. For example, given the information that there is a movie titled "King Kong," we can introduce the following fact rule into the program.

$$movie('King\ Kong') \text{ :- }$$

2.3 Connection Containment

Definition 2.1 (program for a connection) 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, and the input attributes $I(\mathcal{Q})$ and output attributes $O(\mathcal{Q})$. $\Box$

The program $\Pi(\mathcal{Q}_T, \mathcal{V})$ can be constructed from the program $\Pi(\mathcal{Q}, \mathcal{V})$ by removing the connection rules for other connections except T. For instance, Figure 3 shows the program for connection T_1 in Example 1.1. It includes the connection rule for T_1, and the α-rules, and the fact rule in $\Pi(\mathcal{Q}, \mathcal{V})$. That is, $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ can be built by removing the connection rule r_2 from the program $\Pi(\mathcal{Q}, \mathcal{V})$ in Figure 2.

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

Figure 3: The program $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ for connection T_1 in Example 1.1

Definition 2.2 (**answer by a connection**) Let T be a connection in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$, and $\mathcal{D}$ be a database. The *answer by* T, denoted $ans(T, \mathcal{D})$, is the facts of the goal predicate ans that can be computed by evaluating $\Pi(\mathcal{Q}_T, \mathcal{V})$ on $\mathcal{D}$. $\Box$

Definition 2.3 (**connection containment**) 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 resepct to the goal predicate ans, i.e., $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq^{ans} \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$.[2] $\Box$

In other words, if $T_1 \subseteq^{ans} T_2$, then for any database $\mathcal{D}$ of $\mathcal{V}$, we have $ans(T_1, \mathcal{D}) \subseteq ans(T_2, \mathcal{D})$. Thus we can remove T_1 from the query $\mathcal{Q}$, and still compute the same answer to $\mathcal{Q}$ as before.

Consider the two connections in Example 1.1. Now we give an informal proof of $T_1 \subseteq^{ans} T_2$. (The formal proof is in Section 3.) Let $\mathcal{D}$ be a database of $\mathcal{V}$. Assume tuple $<a> \in ans(T_1, \mathcal{D})$, and $<a>$ comes from three tuples: $t_1(disney, m)$ from v_1, $t_2(m, s, w)$ from v_2, and $t_4(s, a)$ from v_4. Since the binding pattern of $v_2(Movie, Star, Award)$ is bbf, and $v_3(B, C)$ is the only view that takes $Star$ as a free attribute, the $Star$ value s in t_2 must be derived from a source query to v_3, which returns a tuple, say, $t_3(m', s)$, whose $Star$ value is s. Similarly, since only $v_1(Studio, Movie)$ takes $Movie$ as a free attribute, the $Movie$ value m' in t_3 must come from a source query to v_1 that returns a tuple, say $t_1'(t', m')$, whose $Movie$ value is m'. Because the binding pattern of $v_1(Studio, Star)$ is bf, and the only $Studio$ value we know is $disney$ (in the initial binding of $\mathcal{Q}$), then $t' = disney$. Since tuples $t_1'(disney, m')$, $t_3(m', s)$, and $t_4(s, a)$ are in views v_1, v_3, and v_4, respectively, $<a>$ is also in the answer by T_2, i.e., $<a> \in ans(T_2, \mathcal{D})$. Therefore, $ans(T_1, \mathcal{D}) \subseteq ans(T_2, \mathcal{D})$, and we have $T_1 \subseteq^{ans} T_2$.

---

[2] A Datalog program Π_1 is *contained in* (*equivalent to*) a Datalog program Π_2 with respect to an IDB predicate F if for all EDBs, the set of F facts computed by Π_1 is a subset of (equivalent to) the set of F facts computed by Π_2.

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 $T_1 \subseteq^{ans} T_2$. We cannot use the existing algorithms for testing containment between CQs [CM77] to solve this problem directly, because $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ and $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ can be recursive Datalog programs, whose containment is known to be undecidable [Shm93]. In the next section we will show that connection containment *is* decidable.

3 Testing Connection Containment

In this section we show that connection containment is decidable, because the containment can be reduced to containment of monadic programs, which is known to be decidable. We also discuss how to test connection containment in some special cases.

3.1 Monadic Datalog 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 the predicate is not on any cycle in the dependency graph of the program [Ull89]. The following theorem is from [CGKV88].

Theorem 3.1 *Containment is decidable for monadic programs.*[3] $\square$

To show that connection containment is decidable, we first give the following proposition:

Proposition 3.1 *Let T be a connection in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$. The program $\Pi(\mathcal{Q}_T, \mathcal{V})$ for T can be translated into an equivalent monadic Datalog program with respect to the goal predicate ans.* $\square$

Proof: Clearly each α-predicate in the program $\Pi(\mathcal{Q}_T, \mathcal{V})$ is nonrecursive, since it is not on any cycle in the dependency graph of $\Pi(\mathcal{Q}_T, \mathcal{V})$. For the connection rule of T in $\Pi(\mathcal{Q}_T, \mathcal{V})$, we replace each α-predicate $\widehat{v_i}$ by the body its α-rule, with the appropriate variable unification. After the substitutions for all the α-predicates, we can remove all the α-rules from $\Pi(\mathcal{Q}_T, \mathcal{V})$ and get a new program Π', which computes the same *ans* facts as $\Pi(\mathcal{Q}_T, \mathcal{V})$ for any database of $\mathcal{V}$. That is, Π' and $\Pi(\mathcal{Q}_T, \mathcal{V})$ are equivalent with respect to the goal predicate *ans*. Program Π' is monadic since its recursive predicates (the domain predicates) are monadic. ■

Consider the program $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ in Figure 3. We replace the IDB predicate $\widehat{v_1}$ in rule r_1 with the body of rule r_3, and substitute variable T with constant *disney*. Similarly, we replace predicate $\widehat{v_2}$ in r_1 with the body of rule r_5, and replace predicate $\widehat{v_4}$ in r_1 with the body of rule r_9, with the appropriate variable unification. We can also remove rule r_7 since the α-predicate $\widehat{v_3}$ is not used by other rules. Figure 4 shows the corresponding monadic program that is equivalent to $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ with respect to the goal predicate *ans*.

[3] *The notion of nonrecursive predicates in [CGKV88] is different from our definition above. It assumes that the nonrecursive predicates do not depend on recursive predicates. That is, a predicate is nonrecursive if it either does not depend on another predicate, or it depends only on nonrecursive predicates. Thus, the program can be unfolded so that nonrecursive predicates do not depend on any other IDB predicate. However, its decidability result of monadic programs can be generalized to our stronger definition of nonrecursive predicates [Var99].*

r'_1: ans(A) :- studio(disney), v_1(disney, M), movie(M), star(S), v_2(M, S, W), star(S), v_4(S, A)
r_4: movie(M) :- studio(T), v_1(T, M)
r_6: award(W) :- movie(M), star(S), v_2(M, S, W)
r_8: star(S) :- movie(M), v_3(M, S)
r_{10}: addr(A) :- star(S), v_4(S, A)
r_{11}: studio(disney) :-

Figure 4: An equivalent program for the program in Figure 3

By Theorem 3.1 and Proposition 3.1, we have the following theorem:

Theorem 3.2 *Connection containment is decidable.* □

[CGKV88] discussed how to test containment of monadic programs using automata theory. It showed that the containment condition can be equivalently phrased as a language-theoretic condition on tree languages. We can use this result to test whether $T_1 \subseteq^{ans} T_2$, where T_1 and T_2 are two connections in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$. Since the decidability of containment of monadic Datalog programs involves a complex algorithm, we introduce the concept of program boundedness. As we will see shortly, when $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ is bounded, we have a much simpler algorithm for testing connection containment.

3.2 Boundedness

A Datalog program is *bounded* if it is possible to eliminate recursions from the program. The following is the formal definition of boundedness [CGKV88].

> Suppose Q is a Datalog program. A database D is a *finite* collection of ground atomic formulas (facts) for the EDB predicates in Q. Let $F_Q^i(D)$ be the collection of facts about an IDB predicate F that can be deduced from D by at most i applications of the rules in Q, and let $F_Q^\infty(D)$ be the collection of facts about F that can be deduced from D by applications of the rules in Q, that is:
>
> $$F_Q^\infty(D) = \bigcup_{i \geq 0} F_Q^i(D)$$
>
> Let $Q^i(D)$ (resp. $Q^\infty(D)$) be the union of $F_Q^i(D)$ (resp. $F_Q^\infty(D)$) for all the IDB predicates F in Q. For each D there is some $k > 0$, depending on D and Q, such that $F_Q^k(D) = F_Q^\infty(D)$. The IDB predicate F is *bounded* in Q if there exists a constant c, depending only on Q, such that for any database D, we have $F_Q^c(D) = F_Q^\infty(D)$. The program is *bounded* if all its IDB predicates are bounded; that is, there exists a constant c, depending only on Q, such that for any database D, we have $Q^c(D) = Q^\infty(D)$.[4]

Note that one *application* of the rules in the program $\Pi(\mathcal{Q}_T, \mathcal{V})$ for a connection T may involve several source queries. For instance, consider connection T_2 in Example 1.1, for the multiple movies

[4]The concept of predicate and program boundedness is different from the concept of attribute boundedness in view adornments.

returned from source S_1, we send several source queries to S_3 (the number of source queries depends on the number of movies from S_1), and these source queries are considered to be one application of the rules in the program $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$.

Since we are more interested in the boundedness of the goal predicate *ans* in the program $\Pi(\mathcal{Q}_T, \mathcal{V})$ for a connection T, in the rest of the paper, we say that program $\Pi(\mathcal{Q}_T, \mathcal{V})$ is bounded (or connection T is bounded) if the goal predicate *ans* in program $\Pi(\mathcal{Q}_T, \mathcal{V})$ is bounded. For instance, as we will see in Section 4, the two connections in Example 1.1 are both bounded. The following example shows an unbounded connection.

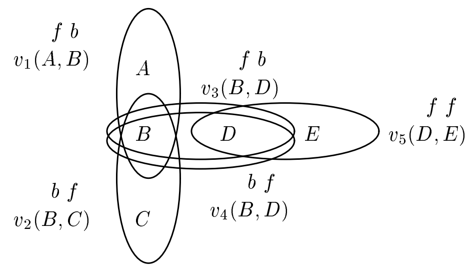

Figure 5: The source descriptions in Example 3.1

EXAMPLE 3.1 Consider the five source views in Figure 5. 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 $\mathcal{Q}$:

$$\mathcal{Q} = <\{A\}, \{C\}, \{T\}>$$

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 6. This program is unbounded, i.e., the goal predicate *ans* 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. For each new B value, it can participate in $v_1 \bowtie v_2$, and generate more answers to the query. (We will give a formal proof of the unboundedness in Section 4.) $\square$

r_1: $\mathtt{ans(C)}$:- $\widehat{\mathtt{v}}_1\mathtt{(a, B)}, \widehat{\mathtt{v}}_2\mathtt{(B, C)}$
r_2: $\widehat{\mathtt{v}}_1\mathtt{(A, B)}$:- $\mathtt{domB(B)}, \mathtt{v}_1\mathtt{(A, B)}$
r_3: $\mathtt{domA(A)}$:- $\mathtt{domB(B)}, \mathtt{v}_1\mathtt{(A, B)}$
r_4: $\widehat{\mathtt{v}}_2\mathtt{(B, C)}$:- $\mathtt{domB(B)}, \mathtt{v}_2\mathtt{(B, C)}$
r_5: $\mathtt{domC(C)}$:- $\mathtt{domB(B)}, \mathtt{v}_2\mathtt{(B, C)}$
r_6: $\widehat{\mathtt{v}}_3\mathtt{(B, D)}$:- $\mathtt{domD(D)}, \mathtt{v}_3\mathtt{(B, D)}$
r_7: $\mathtt{domB(B)}$:- $\mathtt{domD(D)}, \mathtt{v}_3\mathtt{(B, D)}$
r_8: $\widehat{\mathtt{v}}_4\mathtt{(B, D)}$:- $\mathtt{domB(B)}, \mathtt{v}_4\mathtt{(B, D)}$
r_9: $\mathtt{domD(D)}$:- $\mathtt{domB(B)}, \mathtt{v}_4\mathtt{(B, D)}$
r_{10}: $\widehat{\mathtt{v}}_5\mathtt{(D, E)}$:- $\mathtt{v}_5\mathtt{(D, E)}$
r_{11}: $\mathtt{domD(D)}$:- $\mathtt{v}_5\mathtt{(D, E)}$
r_{12}: $\mathtt{domE(E)}$:- $\mathtt{v}_5\mathtt{(D, E)}$
r_{13}: $\mathtt{domA(a)}$:-

Figure 6: The program $\Pi(\mathcal{Q}_T, \mathcal{V})$ in Example 3.1.

3.3 The Case Where Contained Connection is Bounded

Suppose T_1 and T_2 are two connections in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$, and we want to test whether $T_1 \subseteq^{ans} T_2$. If connection T_1 is bounded, then $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ is equivalent to a finite union of CQs. Suppose

$$\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) = \bigvee_{i=1}^{m} Q_i$$

where each Q_i is a CQ. For each $i = 1, \ldots, m$, we test whether Q_i is contained in the program $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ by using the algorithm in [Sag88]. $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ is contained in $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ if and only if all the Q_i's are contained in $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$.

If both $\Pi(\mathcal{Q}_{T_1}, \mathcal{V})$ and $\Pi(\mathcal{Q}_{T_2}, \mathcal{V})$ are bounded, then each of them is equivalent to a finite union of CQs. Suppose $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) = \bigvee_{i=1}^{m} Q_i$, in which each Q_i is a conjunctive query. Similarly, $\Pi(\mathcal{Q}_{T_2}, \mathcal{V}) = \bigvee_{j=1}^{n} P_j$. To test whether $\Pi(\mathcal{Q}_{T_1}, \mathcal{V}) \subseteq^{ans} \Pi(\mathcal{Q}_{T_2}, \mathcal{V})$, we need to test whether the first union of CQs is contained in the second union of CQs. It is known that

$$\bigvee_{i=1}^{m} Q_i \subseteq \bigvee_{j=1}^{n} P_j$$

if and only if for each $i = 1, \ldots, m$, Q_i is contained in some P_j [SY80]. Q_i is contained in P_j if and only if there is a containment mapping from P_j to Q_i [CM77].

EXAMPLE 3.2 The program in Figure 4 can be translated into following CQ:

$$Q\text{: } \mathtt{ans(A)} \text{ :- } \mathtt{v_1(disney, M), v_3(M, S), v_2(M, S, W), v_4(S, A)}$$

Similarly, the program for T_2 in Example 1.1 can be translated into the following CQ:

$$P\text{: } \mathtt{ans(A)} \text{ :- } \mathtt{v_1(disney, M), v_3(M, S), v_4(S, A)}$$

There is a containment mapping from P to Q: $disney \to disney$, $M \to M$, $S \to S$, $A \to A$. Thus $Q \subseteq P$, and $T_1 \subseteq^{ans} T_2$. □

Now we need to solve the problem of testing connection boundedness. [CGKV88] showed that boundedness is decidable for monadic programs, although boundedness in general is undecidable [GMSV93]. The decidability of monadic Datalog programs also involves a complex algorithm that uses tree-automata theory. In Section 4 we develop a polynomial-time algorithm for testing connection boundedness.

4 Connection Boundedness

In this section we give an efficient algorithm for testing connection boundedness. For simplicity, in the rest of the paper, we assume that all the attributes in source views share different domains. However, our results can be easily generalized to the case where multiple attributes share one domain.

We first review some definitions and results from [LC00]. Given a source view v_i, let $\mathcal{B}(v_i)$ and $\mathcal{F}(v_i)$ respectively denote the bound attributes and free attributes in the adorned template of v_i. Let $\mathcal{A}(v_i) = \mathcal{B}(v_i) \cup \mathcal{F}(v_i)$ be all the attributes in v_i. Suppose $\mathcal{W}$ is a set of source views $\mathcal{V}$, let $\mathcal{A}(\mathcal{W})$ denote the attributes in $\mathcal{W}$. For instance, in Example 3.1, $\mathcal{B}(v_1) = \{A\}$, $\mathcal{F}(v_1) = \{B\}$, $\mathcal{A}(v_1) = \{A, B\}$, and $\mathcal{A}(\{v_1, v_2\}) = \{A, B, C\}$.

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 the set of 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 $\mathcal{W}$. For instance, in Example 1.1, *f-closure*$(\{A\}, \{v_1, v_2\}) = \phi$, and *f-closure*$(\{B\}, \{v_1, v_2\}) = \{v_1, v_2\}$.

Let $\mathcal{V}_q$ = *f-closure*$(I(\mathcal{Q}), \mathcal{V})$ be all the *queryable source views*, i.e., the source views that we may eventually query, starting with the initial bindings in $I(\mathcal{Q})$, and perhaps using several preliminary queries to other sources in order to obtain the necessary bindings for these source views. The nonqueryable source views (views in $\mathcal{V} - \mathcal{V}_q$) can be ignored without changing the answer by a connection T, since we cannot retrieve any tuples from them. If there is a nonqueryable view in a connection T, then the answer by the connection is empty.

4.1 Independent Connections

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 1.1 is independent, because *f-closure*$(I(\mathcal{Q}), T_2)$ = *f-closure*$(\{Studio\}, \{v_1, v_3, v_4\}) = T_2$. Connection T_1 is not independent since *f-closure*$(I(\mathcal{Q}), T_1)$ = *f-closure*$(\{Studio\}, \{v_1, v_2, v_4\}) = \{v_1\} \neq T_1$. Similarly, the connection in Example 3.1 is not independent.

Theorem 4.1 *If a connection T is independent, then T is bounded.* □

Proof: Assume $T = \{w_1, \ldots, w_k\}$ is a connection in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$. Since *f-closure*$(I(\mathcal{Q}), T) = T$, there exists a sequence of the views on 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 $\mathcal{D}$ of $\mathcal{V}$, we can compute the answer by T as follows. Compute the corresponding sequence of n *supplementary relations* [BR87, Ull89] $I_1, \ldots, I_n$, where I_i is the supplementary relation after the first i subgoals have been processed. The supplementary relation I_n is the answer by connection T. Therefore, we can compute the answer by the connection using $n+1$ applications of the rules in $\Pi(\mathcal{Q}_T, \mathcal{V})$ (the last application is to evaluate the connection rule for T). ∎

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 3.1.

4.2 BF-chain, BF-loop, and Backward-closure

A sequence of views $w_1, \ldots, w_k$ forms a *BF-chain* (bound-free chain) if for $i = 1, \ldots, k-1$, $\mathcal{F}(w_i) \cap \mathcal{B}(w_{i+1}) \neq \phi$. That is, for two adjacent views w_i and w_{i+1} in the BF-chain, w_i can contribute some bindings to w_{i+1}. The source views w_1 and w_k are the *head* and the *tail* of the BF-chain, respectively. 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 (as shown in Figure 7). For instance, in Figure 5, (v_3, v_4) forms a BF-loop because $\mathcal{F}(v_3) \cap \mathcal{B}(v_4) = \{B\}$ and $\mathcal{F}(v_4) \cap \mathcal{B}(v_3) = \{D\}$.

Suppose A is an attribute in the queryable views $\mathcal{A}(\mathcal{V}_q)$. The *backward-closure of A*, denoted *b-closure*(A), is the set of queryable source 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. 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-

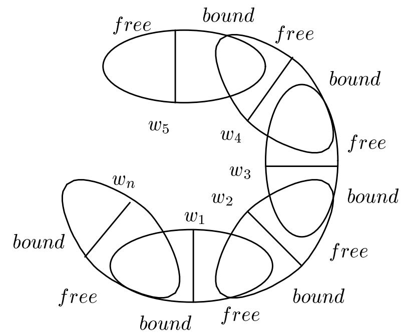

Figure 7: A BF-loop

closures of the attributes in X, i.e., *b-closure*$(X) = \bigcup_{A \in X}$ *b-closure*(A). For instance, in Example 3.1, *b-closure*$(B) = \{v_1, v_3, v_4, v_5\}$, and *b-closure*$(\{B, C\}) = \{v_1, v_2, v_3, v_4, v_5\}$.

Definition 4.1 (**BF-graph**) 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 $\mathcal{F}(v_i) \cap \mathcal{B}(v_j) \neq \phi$. □

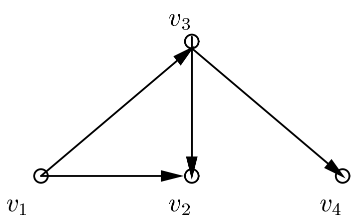

Figure 8: The BF-graph for Example 1.1

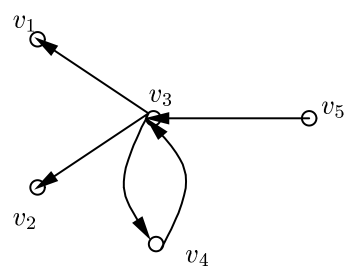

Figure 9: The BF-graph for Example 3.1.

Figures 8 and 9 show the BF-graphs of the source views in Example 1.1 and Example 3.1, respectively. For instance, in Figure 8, 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 among a set of source views if and only if the BF-graph of these views is cyclic.

4.3 Testing Connection Boundedness

Definition 4.2 (**kernel**) Assume T is a connection in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$. A set of attributes $\mathcal{K}$ is a *kernel* of T if

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

and by removing any attribute A from $\mathcal{K}$,

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

□

For instance, in Example 3.1, $\{B\}$ is the only kernel of the connection $\{v_1, v_2\}$. It is shown in [LC00] that if a connection T has different kernels, then they have the same backward-closure. In addition, we can compute the answer by the connection using only the views in *b-closure*$(\mathcal{K}) \cup T$, in which $\mathcal{K}$ is a kernel of T. The following theorem shows that the boundedness of a connection is closely related to the backward-closure of its kernel.

Theorem 4.2 *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 in *b-closure*$(\mathcal{K})$ with distinct views $v_{i_1}, \ldots, v_{i_k}$, 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, 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 by T. Therefore, we can evaluate the rules in finite number of steps to compute the answer by 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 $\Pi(\mathcal{Q}_T, \mathcal{V})$ can we 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, such that $A \in \mathcal{F}(v_k)$. We can add tuples to the relations on the BF-chain, such that only following the BF-chain can we 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 way database $\mathcal{D}$ is constructed, 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 1.1. 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 ϕ, and there is no BF-loop in its backward-closure, thus T_2 is also bounded. In Example 3.1, $\{B\}$ 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 4.2, the connection is unbounded. If a connection T is independent, it has only one kernel, the empty set ϕ. Thus the backward-closure of this kernel is empty, and there is no BF-loop among the views in the backward-closure. By Theorem 4.2, connection T is bounded, which is consistent with Theorem 4.1.

By Theorem 4.2, we give an algorithm called *TestBoundedness* for testing connection boundedness, as shown in Figure 10.

Let us analyze the complexity of the algorithm TestBoundedness. Assume there are n views in $\mathcal{V}$, m views in connection T, k views in $\mathcal{K}$, and p views in *b-closure*$(\mathcal{K})$. [LC00] gives the details how steps 1 to 4 are executed in $O(kn^2)$ time. We can test the cyclicity of the BF-graph G using a depth-first search algorithm in directed graphs, as described in [AHU83]. The complexity of deciding the cyclicity of a directed graph $G(V, E)$ is $O(|E|)$, where E is the set of edges in graph $G(V, E)$.

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 $\mathcal{V}_q = \textit{f-closure}(I(\mathcal{Q}), \mathcal{V})$;
(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 $\textit{b-closure}(\mathcal{K})$;
(5) Build the BF-graph G of $\textit{b-closure}(\mathcal{K})$;
(6) Test the acyclicity of G. If G is acyclic, then T is bounded; otherwise, T is unbounded.

Figure 10: Testing the boundedness of a connection

There could be at most $\binom{p}{2}$ edges in the BF-graph, so step 5 can be done in $O(p^2)$ time, including the time of building the structure of the adjacent vertices for each vertex, as described in [AHU83]. Step 6 can be done in $O(p^2)$ time using a depth-first search algorithm. Therefore, the complexity of the algorithm TestBoundedness is:

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

4.4 Predicate Boundedness

We can extend the result of Theorem 4.2 to the predicate boundedness of the program $\Pi(\mathcal{Q}_T, \mathcal{V})$ for a connection T in a query $\mathcal{Q}$.

Theorem 4.3 *Suppose T is a connection in a query $\mathcal{Q}$ on source descriptions $\mathcal{V}$.*

1. *Assume $domA$ is the domain predicate of an attribute A. Predicate $domA$ is bounded in the program $\Pi(\mathcal{Q}_T, \mathcal{V})$ if and only if there is no BF-loop in b-closure(A).*
2. *Assume v_i is a queryable source view. Predicate $\widehat{v_i}$ is bounded in $\Pi(\mathcal{Q}_T, \mathcal{V})$ if and only if there is no BF-loop in b-closure$(\mathcal{B}(v_i))$.* □

5 Conclusion

In information-integration systems, especially on the World Wide Web, sources have diverse and limited query capabilities. In this paper we studied some open optimization problems in [LC00], including whether the answer computed by one connection in a query is contained in that computed by another connection. We used the result of monadic programs [CGKV88] to show that connection containment is decidable. Since the decidability of monadic Datalog programs involves a complex algorithm, we introduced the question of boundedness for the connections of [LC00]. We developed a polynomial algorithm for testing the boundedness of a connection. When the contained connection is bounded, we can use the algorithms for testing containment of conjunctive queries and unions of conjunctive unions to solve our containment problem.

Acknowledgments: The author thanks Jeff Ullman for his valuable comments on this material.

References

[AHU83] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley Publishing Company, 1983.

[Ama] Amazon.com. `http://www.amazon.com/`.

[Bar] Barnesandnoble.com. `http://www.barnesandnoble.com/`.

[BR87] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 269–283, 1987.

[C+94] Sudarshan S. Chawathe et al. The TSIMMIS project: Integration of heterogeneous information sources. *IPSJ*, pages 7–18, 1994.

[CGKV88] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs. *ACM Symposium on Theory of Computing (STOC)*, pages 477–490, 1988.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *STOC*, pages 77–90, 1977.

[cne] cnet.com. `http://www.cnet.com/`.

[DL97] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI-97*, 1997.

[FLMS99] Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *Proc. of ACM SIGMOD*, pages 311–322, 1999.

[GKD97] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In *Proc. of ACM SIGMOD*, pages 539–542, 1997.

[GMSV93] Haim Gaifman, Harry G. Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *Journal of the ACM*, pages 683–713, 1993.

[HGMN+97] Joachim Hammer, Hector Garcia-Molina, Svetlozar Nestorov, Ramana Yerneni, Markus M. Breunig, and Vasilis Vassalos. Template-based wrappers in the TSIMMIS system. *Proc. of ACM SIGMOD*, pages 532–535, 1997.

[HKWY97] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, pages 276–285, 1997.

[IFF+99] Zachary Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Dan Weld. An adaptive query execution engine for data integration. In *Proc. of ACM SIGMOD*, pages 299–310, 1999.

[LC00] Chen Li and Edward Chang. Query planning with limited source capabilities. *International Conference on Data Engineering (ICDE)*, 2000.

[Li99] Chen Li. Computing complete answers to queries with binding restrictions (extended version). *Technical report, Computer Science Dept., Stanford Univ.,* `http://www-db.stanford.edu/~chenli/pubs/compbp-long.ps`, 1999.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, 1996.

[LYV+98] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey D. Ullman, and Murty Valiveti. Capability based mediation in TSIMMIS. In *Proc. of ACM SIGMOD*, pages 564–566, 1998.

[MW97] David A. Maluf and Gio Wiederhold. Abstraction of representation for interoperation. *International Syposium on Methodologies for Intelligent Systems (ISMIS)*, pages 441–455, 1997.

[myS] mySimon.com. `http://www.mysimon.com/`.

[NS87] Jeffrey F. Naughton and Yehoshua Sagiv. A decidable class of bounded recursions. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 227–236. ACM, 1987.

[PGMW95] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. *International Conference on Data Engineering (ICDE)*, pages 251–260, 1995.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 105–112, 1995.

[Sag88] Yehoshua Sagiv. Optimizing datalog programs. *Foundations of Deductive Databases and Logic Programming*, pages 659–698, 1988.

[Shm93] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

[SY80] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

[TRV98] Anthony Tomasic, Louiqa Raschid, and Patrick Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):808–823, 1998.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes II: The New Technologies.* Computer Science Press, New York, 1989.

[Var99] Moshe Vardi. Personal communication. 1999.

[YLUGM99] Ramana Yerneni, Chen Li, Jeffrey D. Ullman, and Hector Garcia-Molina. Optimizing large join queries in mediation systems. *International Conference on Database Theory (ICDT)*, pages 348–364, 1999.