From: AAAI Technical Report WS-98-14. Compilation copyright © 1998, AAAI (www.aaai.org). All rights reserved.

Query Planning with Disjunctive Sources

Oliver M. Duschka
Socratix Systems, Inc.
oliver@socratix.com*

Michael R. Genesereth
Stanford University
genesereth@cs.stanford.edu

Abstract

We examine the query planning problem in information integration systems in the presence of sources that contain disjunctive information. We show that datalog, the language of choice for representing query plans in information integration systems, is not sufficiently expressive in this case. We prove that disjunctive datalog with inequality is sufficiently expressive, and present a construction of query plans that are guaranteed to extract all available information from disjunctive sources.

1 Introduction

We examine the query planning problem in information integration systems in the presence of sources that contain disjunctive information. The query planning problem in such systems can be formally stated as the problem of answering queries using views (Levy *et al.* 1995; Ullman 1997; Duschka & Genesereth 1997a): View definitions describe the information stored by sources, and query planning requires to rewrite a query into one that only uses these views. In this paper we are going to extend the algorithm for answering queries using conjunctive views that was introduced in (Duschka & Genesereth 1997a) to also be able to handle disjunction in the view definitions.

Example 1 Assume an information source stores flight information. More precisely, the source stores nonstop flights by United Airlines (*ua*) and Southwest Airlines (*sw*), and flights out of San Francisco International Airport (*sfo*) with one stopover. The information stored by this source can be described as being a view over a database with a relation *flight* that stores all nonstop flights. The view definition that describes this source is the following:

$$v(ua,From,To) \;:-\; flight(ua,From,To)$$
$$v(sw,From,To) \;:-\; flight(sw,From,To)$$
$$v(Airline,sfo,To) \;:-\; flight(Airline,sfo,Stopover),\; flight(Airline,Stopover,To)$$

A user might be interested in all cities that have nonstop flights to Los Angeles (*lax*):

$$Q: \quad q(From) \;:-\; flight(Airline,From,lax)$$

If $\langle ua,jfk,lax\rangle$ is a tuple stored by the information source, then there is clearly a nonstop flight from New York (*jfk*) to Los Angeles. On the other hand, if the tuple $\langle ua,sfo,lax\rangle$ is stored by the information source then there doesn't necessarily exist a nonstop flight from San Francisco to Los Angeles. Indeed, this tuple might be stored because there is a flight with one stopover from San Francisco to Los Angeles. The task of query planning in information integration systems is to find a query plan, i.e. a query that only requires views, that extracts as much information as possible from the available sources. All flights to Los Angeles stored by the information source with the exception of flights departing from San Francisco International Airport are nonstop flights. Therefore, the query plan is the following:

$$\mathcal{P}: \quad q(From) \;:-\; v(Airline,From,lax),\; From \neq sfo$$

Note that without the use of the inequality constraint "$From \neq sfo$" it wouldn't be possible to guarantee that all cities returned by the query plan indeed have nonstop flights to Los Angeles. □

Previous work (Duschka & Levy 1997) showed that the expressive power of datalog is both required and sufficient to represent "good" query plans in information integration systems when view definitions are restricted to be conjunctive. As we have seen in Example 1, the presence of disjunctive sources in addition requires the use of inequality constraints in query plans. So far, there are no algorithms that generate query plans with inequality constraints. But the differences between conjunctive sources and disjunctive sources are much deeper. We will see in Example 2 that the expressive power of datalog, even with inequality, is insufficient to represent query plans that extract all available information from disjunctive sources.

Example 2 Assume that there are two information sources available which are described by the following view definitions:

$$v_1(X) :- color(X, red)$$
$$v_1(X) :- color(X, green)$$
$$v_1(X) :- color(X, blue)$$

* Work performed as part of Ph.D. thesis research at Stanford University.

$v_2(X,Y) :- edge(X,Y)$

View v_1 stores vertices that are colored red, green, or blue. View v_2 stores pairs of vertices that are connected by an edge. Assume a user wants to know whether there is a pair of vertices of the same color that are connected by an edge:

$$\mathcal{Q}: \quad q('yes') :- edge(X,Y),\ color(X,Z),\ color(Y,Z).$$

Consider the graphs G_1, G_2, and G_3 in Figure 1. All of these graphs are not three-colorable, i.e. for every possible coloring of the vertices with at most three colors, there will be one edge that connects vertices with the same color. Therefore,

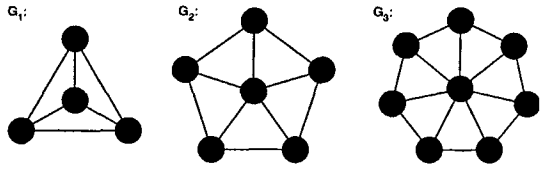

Figure 1: *Examples of graphs that are not 3-colorable.*

every graph that contains G_1, G_2, or G_3 as a subgraph contains an edge that connect two vertices with the same color if the vertices in G_1, G_2, and G_3 are colored by at most three colors. These conditions are easily checkable by a datalog query plan. For example, the query plan $\mathcal{P}_1$ outputs 'yes' exactly when the input graph contains G_1 as a subgraph and when the vertices in G_1 are colored by at most three colors.

$$\begin{aligned}\mathcal{P}_1: \quad q('yes') :- & \ v_1(X_1),\ v_1(X_2),\ v_1(X_3),\ v_1(Y),\\ & v_2(X_1,X_2),\ v_2(X_2,X_3),\\ & v_2(X_3,X_1),\ v_2(X_1,Y),\\ & v_2(X_2,Y),\ v_2(X_3,Y)\end{aligned}$$

It follows that query plan $\mathcal{P}_1$ is contained in query $\mathcal{Q}$. More generally, *every* query plan that checks that the input graph contains a not three-colorable subgraph, and that all the vertices in the subgraph are colored by at most three colors, is contained in $\mathcal{Q}$. Also, every query plan that is contained in $\mathcal{Q}$ outputs *'yes'* only if the graphs described by v_1 and v_2 are not 3-colorable.

It is well known that deciding whether a graph is three-colorable is NP-complete (Karp 1972). Because the problem of evaluating a datalog program has polynomial data complexity (Vardi 1982), this shows that there is *no* datalog query plan that contains *all* the query plans that are contained in $\mathcal{Q}$. Intuitively, the reason is that for every datalog query plan $\mathcal{P}$ that is contained in $\mathcal{Q}$, an additional conjunctive query that tests for one more not three-colorable graph can be added to create a query plan that is still contained in $\mathcal{Q}$, but that is not contained in $\mathcal{P}$. □

Example 2 showed that the expressive power of datalog is insufficient to represent query plans that extract all available information from disjunctive sources. In this paper, we will present a construction of query plans formulated in *disjunctive datalog with inequality* that do guarantee to extract all information. Example 3 shows the query plan resulting from our construction when applied to the query planning problem in Example 2.

Example 3 Let us continue Example 2. The disjunctive datalog query plan that contains all query plans contained in query $\mathcal{Q}$ is the following:

$$\begin{aligned}\mathcal{P}: \quad & q('yes') :- v_2(X,Y),\ c(X,Z),\ c(Y,Z)\\ & c(X,red) \vee c(X,green) \vee c(X,blue) :- v_1(X)\end{aligned}$$

□

1.1 Related work

The problem of answering queries using views (Levy *et al.* 1995) has received considerable attention recently because of its relevance in information integration and data warehousing (Ullman 1997; Levy, Srivastava, & Kirk 1995; Duschka & Genesereth 1997b). Previous algorithms for answering queries using views (Levy, Rajaraman, & Ordille 1996; Kwok & Weld 1996; Duschka & Genesereth 1997a; Qian 1996) deal only with conjunctive view definitions. Sources that can answer an infinite number of conjunctive queries, represented by a datalog program, were studied in (Levy, Rajaraman, & Ullman 1996). To the best of our knowledge, the algorithm presented in this paper is the first one that solves the problem of answering queries using views when view defintions are allowed to contain disjunction.

2 Preliminaries

2.1 Disjunctive datalog

A *disjunctive Horn rule* is an expression of the form

$$p_1(\bar{X}_1) \vee \ldots \vee p_n(\bar{X}_n) :- r_1(\bar{Y}_1),\ \ldots,\ r_m(\bar{Y}_m) \quad (*)$$

where $p_1, \ldots, p_n$, and $r_1, \ldots, r_m$ are predicate names, and $\bar{X}_1, \ldots, \bar{X}_n$, $\bar{Y}_1, \ldots, \bar{Y}_m$ are tuples of variables, constants, and function terms. The *head* of the rule is $p_1(\bar{X}_1) \vee \ldots \vee p_n(\bar{X}_n)$, and its *body* is $r_1(\bar{X}_1), \ldots, r_m(\bar{X}_m)$. Every variable in the head of a rule must also occur in the body of the rule. A *Horn rule* is a disjunctive Horn rule where the head consists of one predicate only. A *(disjunctive) logic query* is a set of (disjunctive) Horn rules, and a *(disjunctive) datalog query* is a set of function-free (disjunctive) Horn rules. If the predicates that appear in the bodies of the rules are allowed to contain the built-in inequality predicate ($\neq$), then the query language is called (disjunctive) datalog *with inequality*. Disjunctive Horn rules with inequality have to satisfy the additional constraint that every variable in the body appears at least once in an uninterpreted predicate. A predicate is an *intensional database predicate*, or *IDB predicate*, in a query $\mathcal{Q}$ if it appears in the head of some rule in $\mathcal{Q}$. Predicates not appearing in any head in $\mathcal{Q}$ are *extensional database predicates*, or *EDB predicates*, in $\mathcal{Q}$. We assume that every query has an IDB predicate q, called the query predicate, that represents the result of $\mathcal{Q}$. A *conjunctive query* is a single non-recursive function-free Horn rule. A *positive query*, also called *disjunctive query*, is a union of conjunctive queries

with the same predicate as head. In this paper, view definitions, abbreviated as $\mathcal{V}$, are sets of positive queries, and user queries are formulated in datalog.

2.2 Semantics

Various semantics have been given to disjunctive datalog queries. The semantic that we are going to present here is commonly known as cautious minimal model semantics (Eiter, Gottlob, & Mannila 1994). As we will see, disjunctive datalog with inequality and cautious minimal model semantics is sufficiently expressive to represent "good" query plans in the presence of disjunctive sources. We will formalize the notion of a "good" query plan in Section 3.

The input of a disjunctive datalog query $\mathcal{Q}$ consists of a database D storing instances of all EDB predicates in $\mathcal{Q}$. A *model* $\mathcal{M}$ of a query $\mathcal{Q}$ and an input database D, denoted as $\mathcal{M} \models \mathcal{Q}(D)$, is an instance of the predicates in $\mathcal{Q}$ such that

(i) $\mathcal{M}$ contains the input database $\mathcal{D}$, and

(ii) whenever there is an instantiation σ of a rule $(*)$ in $\mathcal{Q}$ such that $r_1(\bar{Y}_1)\sigma, \ldots, r_m(\bar{Y}_m)\sigma$ are in $\mathcal{M}$, then $p_i(\bar{X}_n)\sigma$ is in $\mathcal{M}$ for at least one $i \in \{1, \ldots, n\}$.

We denote the instance of the query predicate q in a model $\mathcal{M}$ by $\mathcal{M}_q$. The output of $\mathcal{Q}$, denoted $\mathcal{Q}(D)$, is the largest instance of the query predicate q that occurs in all models of $\mathcal{Q}$ and D, i.e.

$$\mathcal{Q}(D) = \bigcap_{\mathcal{M} : \mathcal{M} \models \mathcal{Q}(D)} \mathcal{M}_q.$$

If $\mathcal{Q}$ is nondisjunctive then there is a unique minimal model $\mathcal{M}$ with $\mathcal{M} \models \mathcal{Q}(D)$ for every D. Then $\mathcal{Q}(D)$ can be determined by, for example, naive bottom-up evaluation (Ullman 1989). For the class of disjunctive queries that we are considering in this paper, there is a direct — although in the worst case co-NP-complete — method to compute $\mathcal{Q}(D)$ using conditional tables (Imielinski & Jr. 1984; Abiteboul & Duschka 1998). However, we are not going to present this evaluation technique here. The central notion for comparing queries is the one of *query containment*. A query $\mathcal{Q}'$ is *contained* in a query $\mathcal{Q}$ if, for all databases D, $\mathcal{Q}'(D)$ is contained in $\mathcal{Q}(D)$.

2.3 Query Plans

A query plan is a query whose EDB predicates are view literals. The expansion $\mathcal{P}^{exp}$ of a query plan $\mathcal{P}$ is a query in which all view literals in $\mathcal{P}$ are replaced by their corresponding view definition. Existentially quantified variables in the view definitions are replaced by new variables in the expansion.

3 Maximal Containment vs. Certain Answers

In this section we are looking more closely at the question of what makes a query plan a "good" query plan. The most basic requirement on a query plan $\mathcal{P}$ is that it produces answers that are asked for in the corresponding query $\mathcal{Q}$ – and nothing else, i.e. that the expansion of $\mathcal{P}$ is contained in $\mathcal{Q}$. Clearly, two query plans $\mathcal{P}_1$ and $\mathcal{P}_2$ can both satisfy this condition, and still $\mathcal{P}_1$ might be better than $\mathcal{P}_2$ because it might be the case that $\mathcal{P}_1$ always produces more answers than $\mathcal{P}_2$, i.e. $\mathcal{P}_2$ is contained in $\mathcal{P}_1$. Previous work therefore focused on the notion of *maximally-contained* query plans.

Definition 1 (maximal containment) Let $\mathcal{L}$ be the language for representing query plans. Given a query $\mathcal{Q}$ and view definitions $\mathcal{V}$, a *maximally-contained* query plan w.r.t. $\mathcal{Q}$, $\mathcal{V}$, and $\mathcal{L}$, denoted by $max_{\mathcal{Q},\mathcal{V},\mathcal{L}}$, is a query plan that contains all query plans whose expansion is contained in $\mathcal{Q}$, i.e.

$$max_{\mathcal{Q},\mathcal{V},\mathcal{L}} \equiv \bigcup_{\mathcal{P} \in \mathcal{L} : \mathcal{P}^{exp} \subseteq \mathcal{Q}} \mathcal{P}.$$

□

Depending on the language used for formulating query plans, maximally-contained query plans might not be guaranteed to exist. For example, if query plans are restricted to be formulated in datalog, then no maximally-contained query plan exists for the query and the view definitions in Example 2. The reason is that there is no (finite) datalog query that is equivalent to the infinite union of conjunctive query plans whose expansion is contained in the user query. Maximally-contained query plans can be found by adding expressive power to the language used to formulate query plans. As seen in Example 3, moving from datalog to disjunctive datalog as the language for query plans is sufficient to represent a maximally-contained query plan.

The notion of maximal containment depends on the concrete language chosen to represent query plans. Indeed, it would be preferable to have a notion of a "good" query plan that is independent from specific languages.

Definition 2 (certain answers) Given a query $\mathcal{Q}$ and view definitions $\mathcal{V}$, the function that computes the set of certain answers w.r.t. $\mathcal{Q}$ and $\mathcal{V}$, denoted by $cert_{\mathcal{Q},\mathcal{V}}$, is the function that maps a view instance to the tuples that are in all results of evaluating $\mathcal{Q}$ on databases consistent with the view instance and the view definitions, i.e. for every instance $\mathcal{I}$ of the views,

$$cert_{\mathcal{Q},\mathcal{V}}(\mathcal{I}) = \bigcap_{D : \mathcal{I} \subseteq \mathcal{V}(D)} \mathcal{Q}(D).$$

□

Maximal containment is a syntactic, proof-theoretic notion. In order to prove that a query plan $\mathcal{P}$ is maximally-contained in a query $\mathcal{Q}$ it is necessary to show that an arbitrarily chosen query plan whose expansion is contained in $\mathcal{Q}$ is also contained in $\mathcal{P}$. On the other hand, the concept of computing certain answers is a semantic, model-theoretic notion. To prove that a function computes all certain answers one has to consider every database that is consistent with the view instance and the view definitions. As in the case of, for example, derivability of a first-order logic formula and its validity, there is also a duality between a query plan being maximally-contained in a query and this query plan computing exactly the certain answers. Lemma 1 formally states this duality.

Lemma 1 *Let $\mathcal{Q}$ be a query, let $\mathcal{V}$ be a set of view definitions, and let $\mathcal{L}$ be a language such that $max_{\mathcal{Q},\mathcal{V},\mathcal{L}}$ exists. Then $cert_{\mathcal{Q},\mathcal{V}}$ is contained in $max_{\mathcal{Q},\mathcal{V},\mathcal{L}}$. Moreover, if $\mathcal{L}$ is monotone then $cert_{\mathcal{Q},\mathcal{V}}$ is equivalent to $max_{\mathcal{Q},\mathcal{V},\mathcal{L}}$.*

Proof. Let $\mathcal{I}$ be an arbitrary view instance. We have to show that

$$\bigcap_{D\,:\,\mathcal{I}\subseteq\mathcal{V}(D)} \mathcal{Q}(D) \;\subseteq\; \bigcup_{\mathcal{P}\in\mathcal{L}\,:\,\mathcal{P}^{exp}\subseteq\mathcal{Q}} \mathcal{P}(\mathcal{I}).$$

Let t be a tuple in $\bigcap_{D\,:\,\mathcal{I}\subseteq\mathcal{V}(D)} \mathcal{Q}(D)$. Consider the following query plan

$$\mathcal{P}: \quad q(t) \;:-\; v_1(t_{11}),\ \ldots,\ v_1(t_{1k_1}),\ \ldots,\ v_n(t_{n1}),\ \ldots,\ v_n(t_{nk_n})$$

where $t_{11},\ldots,t_{1k_1},\ldots,t_{n1},\ldots,t_{nk_n}$ are the tuples in the view instance $\mathcal{I}$. We know that for every database D with $\mathcal{I}\subseteq\mathcal{V}(D)$,

$$\mathcal{P}^{exp}(D) \;\subseteq\; \{t\} \;\subseteq\; \mathcal{Q}(D),$$

and for every database D with $\mathcal{I}\not\subseteq\mathcal{V}(D)$,

$$\mathcal{P}^{exp}(D) \;=\; \mathcal{P}(\mathcal{V}(D)) \;=\; \{\} \;\subseteq\; \mathcal{Q}(D).$$

Therefore, $\mathcal{P}^{exp}\subseteq\mathcal{Q}$. It follows that t is also a tuple in $\bigcup_{\mathcal{P}\in\mathcal{L}\,:\,\mathcal{P}^{exp}\subseteq\mathcal{Q}}\mathcal{P}(\mathcal{I})$.

In order to show equivalence in the case of monotone $\mathcal{L}$, let t be a tuple in $\bigcup_{\mathcal{P}\in\mathcal{L}\,:\,\mathcal{P}^{exp}\subseteq\mathcal{Q}}\mathcal{P}(\mathcal{I})$, and let D be a database with $\mathcal{I}\subseteq\mathcal{V}(D)$. There exists at least one query plan $\mathcal{P}$ with $\mathcal{P}^{exp}\subseteq\mathcal{Q}$ and $t\in\mathcal{P}(\mathcal{I})$. Because of the monotonicity of $\mathcal{P}$ we can conclude that

$$\mathcal{P}(\mathcal{I}) \;\subseteq\; \mathcal{P}(\mathcal{V}(D)) \;\equiv\; \mathcal{P}^{exp}(D) \;\subseteq\; \mathcal{Q}(D).$$

Therefore, tuple t is also in $\bigcap_{D\,:\,\mathcal{I}\subseteq\mathcal{V}(D)}\mathcal{Q}(D)$. □

The following example shows that $max_{\mathcal{Q},\mathcal{V},\mathcal{L}}$ is not guaranteed to be contained in $cert_{\mathcal{Q},\mathcal{V}}$ if query plans are allowed to be nonmonotone.

Example 4 Consider the following view definitions and view instances

$$\begin{array}{lll} \mathcal{V}: & v_1(X) :- p(X) & \mathcal{I}: \quad v_1 = \{a\} \\ & v_2(X) :- p(X) & \quad\;\; v_2 = \{a,b\} \\ & v_2(X) :- r(X) & \end{array}$$

and the following query:

$$\mathcal{Q}: \quad q(X) \;:-\; r(X)$$

The expansion of the nonmonotone query plan

$$\mathcal{P}: \quad q(X) \;:-\; v_2(X),\ \neg v_1(X)$$

is contained in $\mathcal{Q}$. Therefore, $b\in max_{\mathcal{Q},\mathcal{V},\mathcal{L}}(\mathcal{I})$. On the other hand, $b\notin cert_{\mathcal{Q},\mathcal{V}}(\mathcal{I})$ because the database D with $p=\{a,b\}$ and $r=\{\}$ satisfies $\mathcal{I}\subseteq\mathcal{V}(D)$, and $b\notin\mathcal{Q}(D)$. Therefore, $max_{\mathcal{Q},\mathcal{V},\mathcal{L}}$ is not contained in $cert_{\mathcal{Q},\mathcal{V}}$. □

4 Construction of maximally-contained query plans

In this section, we are going to present a construction that produces maximally-contained query plans in the presence of disjunctive sources. As we have seen in Example 2, datalog — and any other language with polynomial data complexity — is not sufficiently expressive to represent maximally-contained query plans in this case. Our construction will therefore produce query plans in a more expressive language, namely disjunctive datalog with inequality.

The central part of our construction is the concept of *disjunctive inverse rules*. Before we can proceed to this definition, we have to define some technical concepts. Let $\mathcal{Q}_1\cup\ldots\cup\mathcal{Q}_n$ be a positive view definition with

$$\begin{array}{ll} \mathcal{Q}_1: & v(\bar{X}_1) \;:-\; p_{11}(\bar{X}_{11}),\ \ldots,\ p_{1m_1}(\bar{X}_{1m_1}) \\ & \ldots \\ \mathcal{Q}_n: & v(\bar{X}_n) \;:-\; p_{n1}(\bar{X}_{n1}),\ \ldots,\ p_{nm_n}(\bar{X}_{nm_n}). \end{array}$$

We can assume without loss of generality that the sets of variables $\bar{X}_1,\ldots,\bar{X}_n$ are all mutually disjoint. Given a tuple t in an instance of v, we have to determine which of the conjunctive queries $\mathcal{Q}_1,\ldots,\mathcal{Q}_n$ might have generated t. If there is a tuple t such that t can be generated by any of the queries $\mathcal{Q}_{i_1},\ldots,\mathcal{Q}_{i_k}$, then these queries are called *truly disjunctive*. More formally, queries $\mathcal{Q}_{i_1},\ldots,\mathcal{Q}_{i_k}$ are called truly disjunctive if there is a substitution σ such that $\bar{X}_{i_1}\sigma=\bar{X}_{i_2}\sigma=\ldots=\bar{X}_{i_k}\sigma$. $\bar{X}_{i_1}\sigma$ is a *witness* of $\mathcal{Q}_{i_1},\ldots,\mathcal{Q}_{i_k}$ being truly disjunctive.

Let the arity of v be α, and let $\pi_1,\ldots,\pi_\alpha$ be new constants. A conjunction of inequalities φ involving only the new constants $\pi_1,\ldots,\pi_\alpha$ and the constants in $\bar{X}_1,\ldots,\bar{X}_n$ is called an *attribute constraint*. A conjunctive query $\mathcal{Q}_i$ *satisfies* an attribute constraint φ if all inequalities in φ hold after replacing each π_j in φ by the corresponding $\bar{X}_i[j]$. If queries $\mathcal{Q}_{i_1},\ldots,\mathcal{Q}_{i_k}$ are truly disjunctive with most general witness $\bar{Z}$, and there is an attribute constraint φ satisfied by $\bar{X}_{i_1},\ldots,\bar{X}_{i_k}$, but not satisfied by any $\bar{X}_j$ with $j\in\{1,\ldots,n\}-\{i_1,\ldots,i_k\}$ and $\bar{X}_j$ unifiable with $\bar{Z}$, then φ is called an *exclusion condition* for $\mathcal{Q}_{i_1},\ldots,\mathcal{Q}_{i_k}$.

Example 5 Let us continue Example 1. The following is the positive view definition we considered there with head variables renamed appropriately:

$$\begin{array}{ll} \mathcal{Q}_1: & v(ua,F_1,T_1) \;:-\; flight(ua,F_1,T_1) \\ \mathcal{Q}_2: & v(sw,F_2,T_2) \;:-\; flight(sw,F_2,T_2) \\ \mathcal{Q}_3: & v(A_3,sfo,T_3) \;:-\; flight(A_3,sfo,S),\ flight(A_3,S,T_3) \end{array}$$

Here is a list of truly disjunctive queries together with their most general witness and their most general exclusion condition:

$\mathcal{Q}_1$	$\langle ua,F_1,T_1\rangle$	$\pi_2\neq sfo$
$\mathcal{Q}_2$	$\langle sw,F_2,T_2\rangle$	$\pi_2\neq sfo$
$\mathcal{Q}_3$	$\langle A_3,sfo,T_3\rangle$	$\pi_1\neq ua\ \&\ \pi_1\neq sw$
$\mathcal{Q}_1,\mathcal{Q}_3$	$\langle ua,sfo,T_1\rangle$	*true*
$\mathcal{Q}_2,\mathcal{Q}_3$	$\langle sw,sfo,T_2\rangle$	*true*

This list tells us that a tuple of the form $\langle ua,F,T\rangle$, for example, with $F\neq sfo$ must have been generated by query $\mathcal{Q}_1$, and

a tuple of the form (sw,sfo,T) must have been generated by either query Q_2 or query Q_3. □

We are now able to define the central concept of disjunctive inverse rules. Intuitively, inverse rules describe all the databases that are consistent with the view definitions given a specific view instance.

Definition 3 (Disjunctive inverse rules) Let

$$\begin{aligned} Q_1: \quad & v(\bar{X}_1) \ :- \ p_{11}(\bar{X}_{11}), \ \ldots, \ p_{1m_1}(\bar{X}_{1m_1}) \\ & \ldots \\ Q_n: \quad & v(\bar{X}_n) \ :- \ p_{n1}(\bar{X}_{n1}), \ \ldots, \ p_{nm_n}(\bar{X}_{nm_n}) \end{aligned}$$

be a positive view definition with disjoint sets of head variables $\bar{X}_1, \ldots, \bar{X}_n$, and variables $X_1, \ldots, X_s$ in the bodies but not in $\bar{X}_1, \ldots, \bar{X}_n$. Let $f_1, \ldots, f_s$ be new function symbols. Then for every set of truly disjunctive queries $Q_{i_1}, \ldots, Q_{i_k}$ with most general witness $\bar{Z}$ and most general exclusion condition φ, the following rules are *disjunctive inverse rules*:

$$p_{i_1\delta_1}(\bar{X}'_{i_1\delta_1}) \vee \ldots \vee p_{i_k\delta_k}(\bar{X}'_{i_k\delta_k}) \ :- \ v(\bar{Z}), \ \varphi'$$

with $\delta_l \in \{1, \ldots, m_l\}$ for $l = 1, \ldots, k$, and

$$\bar{X}'_{\beta\gamma}[j] = \begin{cases} \bar{Z}[j'] & : \text{ if } (\bar{X}_{\beta\gamma})[j] = \bar{X}_\beta[j'] \text{ for some } j' \\ \bar{X}_{\beta\gamma}[j] & : \text{ if } (\bar{X}_{\beta\gamma})[j] \text{ is a constant} \\ f_\kappa(\bar{Z}) & : \text{ if } (\bar{X}_{\beta\gamma})[j] = X_\kappa \end{cases}$$

for all β, γ, j. Condition φ' is generated from φ by replacing each constant π_j in φ by the corresponding variable or constant $\bar{Z}[j]$. □

We denote the set of disjunctive inverse rules of a set $\mathcal{V}$ of view definitions by $\mathcal{V}^{-1}$.

Example 6 The disjunctive inverse rules of the positive view definition in Example 5 are the following rules:

$$\begin{aligned} & flight(ua,F_1,T_1) \ :- \ v(ua,F_1,T_1), \ F_1 \neq sfo \\ & flight(sw,F_2,T_2) \ :- \ v(sw,F_2,T_2), \ F_2 \neq sfo \\ & flight(A_3,sfo,f(A_3,sfo,T_3)) \ :- \ v(A_3,sfo,T_3), \\ & \qquad\qquad A_3 \neq ua, \ A_3 \neq sw \\ & flight(A_3,f(A_3,sfo,T_3),T_3) \ :- \ v(A_3,sfo,T_3), \\ & \qquad\qquad A_3 \neq ua, \ A_3 \neq sw \\ & flight(ua,sfo,T_1) \vee flight(ua,sfo,f(ua,sfo,T_1)) \\ & \qquad\qquad :- \ v(ua,sfo,T_1) \\ & flight(ua,sfo,T_1) \vee flight(ua,f(ua,sfo,T_1),T_1) \\ & \qquad\qquad :- \ v(ua,sfo,T_1) \\ & flight(ua,sfo,T_1) \vee flight(sw,sfo,f(sw,sfo,T_2)) \\ & \qquad\qquad :- \ v(sw,sfo,T_2) \\ & flight(ua,sfo,T_1) \vee flight(sw,f(sw,sfo,T_2),T_2) \\ & \qquad\qquad :- \ v(sw,sfo,T_2) \end{aligned}$$

□

In the following we will consider the query plan consisting of the rules of a datalog query Q together with the disjunctive inverse rules $\mathcal{V}^{-1}$. Disjunctive inverse rules contain function symbols. Therefore, the output of a query plan $Q \cup \mathcal{V}^{-1}$ can contain tuples with function symbols. Given a query plan $\mathcal{P}$ and an instance $\mathcal{I}$, let us denote by $\mathcal{P}(\mathcal{I})\downarrow$ the subset of $\mathcal{P}(\mathcal{I})$ that doesn't contain function symbols. As shown in (Duschka & Genesereth 1997a) for datalog query plans, it is possible to transform a query plan of the form $Q \cup \mathcal{V}^{-1}$ into a datalog query plan, denoted as $(Q \cup \mathcal{V}^{-1})\downarrow$, that computes only the tuples without function symbols, i.e.

$$(Q \cup \mathcal{V}^{-1})(\mathcal{I})\downarrow \ = \ (Q \cup \mathcal{V}^{-1})\downarrow(\mathcal{I})$$

for all instances $\mathcal{I}$. This transformation can easily be generalized to disjunctive datalog query plans.

The following theorem shows that the query plan $(Q \cup \mathcal{V}^{-1})\downarrow$ is guaranteed to be maximally-contained in Q. The proof of the theorem crucially uses the duality between maximal containment and certain answers discussed in Section 3.

Theorem 1 *For every datalog query Q and every set of positive view definitions $\mathcal{V}$, the disjunctive datalog query plan $(Q \cup \mathcal{V}^{-1})\downarrow$ is maximally-contained in Q.*

Proof. (sketch) Let $\mathcal{I}$ be a view instance. Because the Q part of query plan $Q \cup \mathcal{V}^{-1}$ does not contain any EDB predicates, and because all the predicates in the bodies of $\mathcal{V}^{-1}$ are EDB predicates, every bottom-up evaluation of $Q \cup \mathcal{V}^{-1}$ necessarily first has to evaluate $\mathcal{V}^{-1}$ before evaluating Q. Therefore,

$$(Q \cup \mathcal{V}^{-1})(\mathcal{I}) \ = \ \bigcap_{\mathcal{M} : \mathcal{M} \models \mathcal{V}^{-1}(\mathcal{I})} Q(\mathcal{M}).$$

Since disjunctive datalog queries are monotone, it suffices by Lemma 1 to show that

$$\underbrace{\Big(\bigcap_{\mathcal{M} : \mathcal{M} \models \mathcal{V}^{-1}(\mathcal{I})} Q(\mathcal{M}) \Big)\downarrow}_{A} \ = \ \underbrace{\bigcap_{D : \mathcal{I} \subseteq \mathcal{V}(D)} Q(D)}_{B}.$$

Let $\mathcal{M}$ be a model of $\mathcal{V}^{-1}$ and $\mathcal{I}$. By the construction of $\mathcal{V}^{-1}$ we know that $\mathcal{I} \subseteq \mathcal{V}(\mathcal{M})$. Therefore, $B \subseteq A$. Because B doesn't contain function symbols it follows that $B \subseteq A\downarrow$.

Let D be a database with $\mathcal{I} \subseteq \mathcal{V}(D)$. Consider all the models of $\mathcal{V}^{-1}$ and $\mathcal{V}(D)$. One of these models coincides with D with the only difference that some function symbols in the model are replaced by constants in D. Let $\mathcal{M}$ be this model, and let t be a tuple without function symbols in $\mathcal{M}_q$. Because datalog queries are monotone when constants in the input database are made equal, it follows that $Q(\mathcal{M})\downarrow \subseteq Q(D)$. Therefore, $A\downarrow \subseteq B$. □

Theorem 2 *For every datalog query Q and every set of positive view definitions $\mathcal{V}$, the disjunctive datalog query plan $(Q \cup \mathcal{V}^{-1})\downarrow$ can be evaluated in co-NP time (data complexity).*

Proof. Let t be a tuple that is *not* in $(Q \cup \mathcal{V}^{-1})\downarrow(\mathcal{I})$ for some instance $\mathcal{I}$. Then there is some model $\mathcal{M}$ of $\mathcal{V}^{-1}$ and $\mathcal{I}$ such that t is not in $Q(\mathcal{M})$. If $\mathcal{I}$ contains n tuples and the longest conjunct in $\mathcal{V}$ has m literals, then there is a submodel $\mathcal{M}'$ of $\mathcal{M}$ with at most $n \times m$ tuples that is still a model of $\mathcal{V}^{-1}$. Because of the monotonicity of Q, t is also not in $\mathcal{M}'$. Moreover, checking that $\mathcal{M}'$ is a model of $\mathcal{V}^{-1}$, and that t is not in $Q(\mathcal{M})$ can be done in polynomial time. □

5 Conclusions and future work

We considered the problem of answering queries using views with positive view definitions. We showed that datalog is not expressive enough to represent maximally-contained query plans in this case. One the other hand, disjunctive datalog with inequality *is* expressive enough. We presented a construction of maximally-contained query plans in this more expressive language.

The data complexity of evaluating disjunctive datalog queries with inequality in general is co-NP-complete. However, it seems like there are subcases that might allow polynomial time evaluation. The following subcases are likely candidates: (i) $\mathcal{Q}$ has no projections, (ii) $\mathcal{Q}$ is conjunctive and $\mathcal{V}$ has no projections, and (iii) all view definitions in $\mathcal{V}$ have at most two disjuncts. Future work needs to be devoted to look more closely at these subcases.

Acknowledgments

We would like to thank Serge Abiteboul for pointing out the relationship between maximal containment and the computation of certain answers. Also, thanks to Yehoshua Sagiv and Werner Nutt for helpful discussions on this topic.

References

Abiteboul, S., and Duschka, O. M. 1998. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM Symposium on Principles of Database Systems, PODS '98.*

Duschka, O. M., and Genesereth, M. R. 1997a. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems, PODS '97*, 109 – 116.

Duschka, O. M., and Genesereth, M. R. 1997b. Infomaster — an information integration tool. In *Proceedings of the International Workshop on Intelligent Information Integration during the 21st German Annual Conference on Artificial Intelligence, KI-97.*

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

Eiter, T.; Gottlob, G.; and Mannila, H. 1994. Adding disjunction to datalog. In *Proceedings of the Thirteenth ACM Symposium on Principles of Database Systems*, 267 – 278.

Imielinski, T., and Jr., W. L. 1984. Incomplete information in relational databases. *J. ACM* 31(4):761–791.

Karp, R. M. 1972. Reducibility among combinatorial problems. *Complexity of Computer Computations* 85 – 104.

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

Levy, A. Y.; Mendelzon, A. O.; Srivastava, D.; and Sagiv, Y. 1995. Answering queries using views. In *Proceedings of the 14th ACM Symposium on Principles of Database Systems.*

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases*, 251–262.

Levy, A. Y.; Rajaraman, A.; and Ullman, J. D. 1996. Answering queries using limited external processors. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems.*

Levy, A. Y.; Srivastava, D.; and Kirk, T. 1995. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems: Integrating Artificial Intelligence and Database Technologies* 5(2):121–43.

Qian, X. 1996. Query folding. In *Proceedings of the 12th International Conference on Data Engineering*, 48–55.

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

Ullman, J. D. 1997. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory.*

Vardi, M. Y. 1982. The complexity of relational query languages. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 137 – 146.