Answering Queries Using Materialized Views with Disjunctions*

Foto N. Afrati[1], Manolis Gergatsoulis[2], and Theodoros Kavalieros[1]

[1] Dept. of Electrical and Computer Engineering,
National Technical University of Athens,
157 73 Athens, Greece,
{afrati,kavalier}@softlab.ece.ntua.gr
[2] Inst. of Informatics & Telecom.,
N.C.S.R. 'Demokritos',
153 10 A. Paraskevi Attikis, Greece
manolis@iit.demokritos.gr

Abstract. We consider the problem of answering datalog queries using materialized views. More specifically, queries are rewritten to refer to views instead of the base relations over which the queries were originally written. Much work has been done on program rewriting that produces an equivalent query. In the context of information integration, though, the importance of using views to infer as many answers as possible has been pointed out. Formally, the problem is: Given a datalog program $\mathcal{P}$ is there a datalog program $\mathcal{P}_v$ which uses only views as EDB predicates and (i) produces a subset of the answers that $\mathcal{P}$ produces and (ii) any other program $\mathcal{P}'_v$ over the views with property (i) is contained in $\mathcal{P}_v$? In this paper we investigate the problem in the case of disjunctive view definitions.

1 Introduction

A considerable amount of recent work has focused on using materialized views to answer queries [1, 7, 5, 12, 16, 13, 4]. This issue may arise in several situations, e.g., if the relations mentioned in the query are not actually stored or are impossible to consult or are very costly to access. The ability to use views is important in many applications, including information integration, where information sources are considered to store materialized views over a global database schema [2, 19, 14].

Suppose that we want to integrate three databases that provide flight information. These three databases can be seen as views over the EDB predicates $international\_flight(X, Y)$ and $local\_flight(X, Y)$ (which have the meaning that there is a non-stop international flight from a greek city X to city Y and,

---

* This work has been partially supported by the Greek General Secretariat of Research and Technology under the project "Logic Programming Systems and Environments for developing Logic Programs" of ΠENEΔ'95, contract no 952.

there is a non-stop local flight from a greek city X to a greek city Y respectively):

$$\begin{aligned}
&v_1(X) :- international\_flight(Athens, X)\\
&v_1(X) :- international\_flight(Rhodes, X)\\
&v_2(X,Y) :- local\_flight(X, Athens), local\_flight(Athens, Y)\\
&v_3(X) :- local\_flight(X, Rhodes)
\end{aligned}$$

Suppose that the user is interested in whether there is a way to fly from a greek city X to a city Y in another country making at most one intermediate connection in a greek city and then a direct international flight i.e. the following query $\mathcal{P}$, is asked:

$$\begin{aligned}
&p(X,Y) :- international\_flight(X,Y)\\
&p(X,Y) :- local\_flight(X,Z), international\_flight(Z,Y)
\end{aligned}$$

The user does not have access to the databases that contain the facts in the predicates *international_flight* and *local_flight* but only to the views. Using only the materialized views v_1, v_2 and v_3, it is not possible to find all the answers. The best one can do is to retrieve all the flights that use either the international airport of *Athens* or the international airport of *Rhodes*, i.e. ask the following query $\mathcal{P}_v$, on the available data:

$$\begin{aligned}
&p(Athens, Y) :- v_2(W, Rhodes), v_1(Y)\\
&p(Rhodes, Y) :- v_2(Rhodes, W), v_1(Y)\\
&p(Athens, Y) :- v_3(Athens), v_1(Y)\\
&p(X,Y) :- v_3(X), v_2(X, W), v_1(Y)
\end{aligned}$$

In fact, the program $\mathcal{P}_v$ which is said to be a *retrievable program* (i.e., it contains only the view predicates as EDB predicates), is maximal in the sense that any datalog program which uses only the views v_1, v_2, v_3 as EDB predicates and produces only *correct* answers is contained in $\mathcal{P}$. Such a program is said to be a *retrievable maximally contained program.*

The problem that we consider here is : Given a datalog query and views defined over its EDB predicates, is there a retrievable maximally contained datalog program?

Previous work on this problem is done in [7, 12] where they restricted views to being defined by conjunctive queries, whereas in [8, 6] disjunctive view definitions are considered and is shown how to express a retrievable maximally contained program in disjunctive datalog with inequality. In [1], the problem of computing correct (called *certain*) answers is considered, also, in the general case where views are defined by datalog programs; they investigate the data complexity problem under both the closed world assumption and the open world assumption. In this paper, we investigate the case where views have disjunctions in their definition. Here we prove the following results:

a) In the case where both the query and the views are given by non-recursive datalog programs, we identify a non-trivial class of instances of the problem where there exists a retrievable datalog($\neq$) program maximally contained in the query; in this case, we give an algorithm to obtain it; this program computes *all* correct answers. We know, though, that, in general such a program does not exist [1].

b) When the views are defined by recursive datalog programs and the query by non-recursive datalog program, we reduce the problem to that of non-recursive view definitions.

c) In the case the query is given by a recursive datalog program and the views by non-recursive datalog programs, we construct a simple disjunctive logic program, which, applied on a view instance, computes all correct answers whenever it halts. This result is also obtained in [8] under a similar technique.

d) Finally, when both the views and the query are given by recursive programs, we prove that it is undecidable whether there exists a non-empty retrievable program $\mathcal{P}_v$. Still, we give a criterion that gives a negative answer, in some special cases. In fact, the undecidability result holds even in the case of simple chain programs [20].

Other related work is done in [5, 16, 13, 4], where the approaches used essentially look for rewritings producing equivalent queries. It is known [13] that the problem of finding a rewriting that produces an equivalent program is NP-complete, in the case, where both the query and the views are defined by disjunctions of conjunctive queries. Other work concern conjunctive queries (the views and the question asked) and include [13, 16] where they give nondeterministic polynomial time algorithms to find either a single retrievable query equivalent to the given query or a set of retrievable queries whose union is contained in the given query Q and that contains any other retrievable datalog query that is contained in Q. Special classes of conjunctive queries are examined in [5].

2 Preliminaries

2.1 Logic programs and datalog

A *disjunctive clause* C is a formula of the form [15]

$$A_1 \vee \ldots \vee A_m : -B_1, \ldots, B_n$$

where $m \geq 1$, $n \geq 0$ and $A_1, \ldots, A_m, B_1, \ldots, B_n$ are atoms. If $n = 0$, C is called a *positive disjunctive clause.* If $m = 1$, C is called a *definite clause* or *Horn clause.* A *definite/disjunctive program* is a finite set of definite/disjunctive clauses. A *Datalog program* is a set of function free Horn clauses. A *Datalog($\neq$) program* is a set of function free Horn clauses whose bodies are allowed to contain atoms whose predicate is the built-in inequality predicate ($\neq$). The left hand side of a rule is called the *head* of the rule and the right hand side is the *body* of the rule. A predicate is an *intensional database predicate*, or *IDB predicate*, in a program $\mathcal{P}$ if it appears at the head of a rule in $\mathcal{P}$, otherwise it is an *extensional database (EDB) predicate.* A *conjunctive query* is a single non-recursive function-free Horn rule.

Let D be a finite set. A *database* over domain D is a finite relational structure $\mathcal{D} = (D, r_1, \ldots, r_n)$, where each r_i is a relation over D; a *substructure* of a database is a relational structure $(D', r'_1, \ldots, r'_n)$, where each r'_i contains a subset of the facts contained in r_i.

A *query* is a function from databases (of some fixed type) to relations of fixed arity over the same domain; it has to be *generic*, i.e., invariant under renamings of the domain.

Datalog programs may be viewed as a declarative query language with the following semantics. Let $\mathcal{D}$ be a database, thought of as a collection of facts about the EDB predicates of a program $\mathcal{P}$. Let $Q^k_{\mathcal{P},p}(\mathcal{D})$ be the collection of facts about an IDB predicate p that can be deduced from $\mathcal{D}$ by at most k applications of the rules in $\mathcal{P}$. If we consider p the *goal* predicate (or *output* predicate), then $\mathcal{P}$ expresses a query $Q_{\mathcal{P},p}$, where

$$Q_{\mathcal{P},p}(\mathcal{D}) = \bigcup_{k \geq 0} Q^k_{\mathcal{P},p}(\mathcal{D})$$

We will sometimes write $Q_{\mathcal{P}}$ or $\mathcal{P}_p(\mathcal{D})$ instead of $Q_{\mathcal{P},p}$.

The above definition also gives a particular algorithm to compute $Q_{\mathcal{P},p}$: initialize the IDB predicates to be empty, and repeatedly apply the rules to add tuples to the IDB predicates, until no new tuples can be added. This is known as *bottom-up evaluation.* A *derivation tree* is a tree depicting the bottom up evaluation of a specific derived fact in the IDB. Its nodes are labeled by facts and each node with its children correspond to an instantiation of a rule in $\mathcal{P}$ [18].

Given a datalog program, we can define a *dependency graph*, whose nodes are the predicate names appearing in the rules. There is an edge from the node of predicate p to the node of predicate p' if p' appears in the body of a rule whose head predicate is p. The program is *recursive* if there is a cycle in the dependency graph. We define, also the *adorned dependency graph* as follows: We assign labels (possible none or more than one) on each edge: Suppose there is a rule with head predicate symbol p and predicate symbol p' appears somewhere in the body; suppose that the i-th argument of p is the same variable as the j-th argument of p'. Then, on edge $e = (p, p')$ we assign a label $l =i \rightarrow j$; we call the pair (e, l) a *p-pair.* We define a *perfect cycle* in the adorned dependency graph to be a sequence of p-pairs $(e_1, l_1), \ldots, (e_k, l_k)$ such that the edges $e_1, \ldots, e_k$ form a cycle and, moreover, the following holds: for all $q = 1, \ldots, k-1$ if $l_q =i \rightarrow j$ then $l_{q+1} =j \rightarrow j'$ for some j' and, $l_k =i \rightarrow j$ then $l_1 =j \rightarrow j'$ for some j'. We say that a datalog program *has no presistencies* iff there is no perfect cycle in the adorned dependency graph.

Given any datalog program $\mathcal{P}$, we can unwind the rules several times and produce a conjunctive query (over the EDB predicates of $\mathcal{P}$) generated by $\mathcal{P}$. Program $\mathcal{P}$ can, therefore, be viewed as being equivalent to an infinite disjunction of all these conjunctive queries. Considering a conjunctive query, we freeze the body of the query by turning each of the subgoals into facts in a database; we obtain, thus, the *canonical database* of this query. The constants in the canonical database that correspond to variables that also appear in the head of the query are conveniently called *head constants*. If all EDB predicates are binary, we may view a canonical database as a labeled graph. Correspondingly, we can refer to a canonical database *generated* by a datalog program. We say that a datalog program is *connected* iff all head variables of the program rules occur also in the bodies of the corresponding rules and any canonical database (viewed as a hypergraph) is connected.

A *containment mapping* from a conjunctive query Q_1 to a conjunctive query Q_2, is a function, h, that maps all variables of Q_1 on variables of Q_2, so that whenever an atom $p(X_1, X_2, \ldots)$ occurs in Q_1, then an atom $p(h(X_1), h(X_2), \ldots)$ occurs in Q_2.

The following are technical lemmata used in the proof of the main results.

Lemma 1. *Let $\mathcal{P}$ be a Datalog program and c a conjunctive query. There exists a positive integer N depending only on the sizes of the program and the query so that the following holds: If there is a conjunctive query c_i generated by $\mathcal{P}$ such that there is a containment mapping from c to c_i, then there is a conjunctive query c_j generated by $\mathcal{P}$, of size $< N$, such that there is a containment mapping from c to c_j.*

Proof. First, we note, that, given a sufficiently large conjunctive query generated by a datalog program, we can apply a pumping on it, obtaining a shorter conjunctive query generated by the same program (see [3]).

Suppose c_i is of size $> N$. The containment mapping from c to c_i maps c on at most $|c|$ constants in c_i. Thus, there are $> N - |c|$ constants in c_i that are not involved in this mapping. Consider this "free" part of c_i and do on it a pumping, obtaining, thus a shorter conjunctive query, on which the containment mapping is preserved. $\square$

Definition 1. *Consider a database. A* neighborhood of size k *in the database is a connected substructure of size k. If we view the database as the canonical database of a conjunctive query, then, we can refer to neighborhoods in a conjunctive query accordingly.*

Lemma 2. *Suppose two conjunctive queries Q_1, Q_2 such that: each neighborhood of size up to k appears in one query iff it appears in the other. Then, for any conjunctive query Q of size up to k, there is containment mapping from Q to the query Q_1 iff there is containment mapping from Q to Q_2.*

2.2 Retrievable programs

Let $\mathcal{P}$ be any datalog program over EDB predicates $e_1, e_2, \ldots, e_m$ and with output predicate q. Let $v_1, v_2, \ldots, v_n$ be views defined by datalog programs over EDB predicates $e_1, e_2, \ldots, e_m$; call $\mathcal{P}_{\mathcal{V}}$ the union of programs that define the views. We want to answer the query $Q_{\mathcal{P}}$ assuming that we do not have access to database relations $e_1, e_2, \ldots, e_m$, but only to views $v_1, \ldots, v_n$; we need a "certain" answer no matter which is the database that yielded the given view instance.

Definition 2. *A* certain *answer to query $Q_{\mathcal{P}}$, given a view instance $\mathcal{D}_{\mathcal{V}}$, is a tuple t such that: for each database $\mathcal{D}$ over $e_1, e_2, \ldots, e_m$ with $\mathcal{D}_{\mathcal{V}} \subseteq \mathcal{P}_{\mathcal{V}}(\mathcal{D})$, t belongs to $Q_{\mathcal{P}}(\mathcal{D})$.*

The above definition assumes the open world assumption; under the closed world assumption, the definition requires that $\mathcal{D}_{\mathcal{V}} = \mathcal{P}_{\mathcal{V}}(\mathcal{D})$; to compute a certain answer in this case is harder (see [1]).

Let $\mathcal{P}_v$ be a datalog program over EDB predicates $v_1, v_2, \ldots, v_n$. We say that $\mathcal{P}_v$ *is a retrievable program contained* in $\mathcal{P}$ if, for each view instance $\mathcal{D}_{\mathcal{V}}$, all tuples in $\mathcal{P}_v(\mathcal{D}_{\mathcal{V}})$ are certain answers.

Example 1. Let views v_1, v_2 be defined over the EDB predicates e_1, e_2, e_3, e_4:

$$\begin{aligned} v_1(X,Y) &:- \ e_1(X,Y) \\ v_1(X,Y) &:- \ e_1(X,Z), e_2(Z,Y) \\ v_2(X,Y) &:- \ e_3(X,Y), e_4(Y) \end{aligned}$$

and the query $\mathcal{P}$ by:

$$p(X) :- \ e_3(Y,X), e_1(X,Z)$$

Then a retrievable program maximally contained in $\mathcal{P}_v$, is:

$$p_v(X) :- \ v_2(Y,X), v_1(X,Z)$$

If $\mathcal{P}_v$ is applied on an instance of v_1, v_2, it will produce a set of answers; this is a subset of the answers which will be produced if query $\mathcal{P}$ is applied on some instance of the relations e_1, e_2, e_3, e_4 which yields the given instance of v_1, v_2. For example, suppose $v_1 = \{(a,b),(b,c),(b,d),(c,f)\}$ and $v_2 = \{(d,a)\}$. Then $\mathcal{P}_v$ will produce the set of answers $\{(a)\}$. A database that yields the above view instance could have been either $\mathcal{D}_1 = \{e_1(a,b), e_1(b,c), e_1(c,f),\ e_2(c,d), e_3(d,a),\ e_3(b,c),\ e_4(a)\}$ or $\mathcal{D}_2 = \{e_1(a,b), e_1(b,d),\ e_1(c,f), e_2(d,c),\ e_2(f,g),\ e_3(d,a),\ e_3(d,b),\ e_4(a),\ e_4(c),\ e_4(b)\}$ (or infinitely many other alternatives). If we apply $\mathcal{P}$ on $\mathcal{D}_1$, we get the answers $\{(a),(c)\}$, whereas if we apply $\mathcal{P}$ on $\mathcal{D}_2$ we get the answers $\{(a),(b)\}$; tuple (a) is the only answer common to both. It turns out that, for any view instance, program $\mathcal{P}_v$ provably derives exactly those answers that are common to all relational structures over e_1, e_2, e_3, e_4 that yield this view instance. Note, that if the programs that define the views are applied on $\mathcal{D}_2$, then more tuples will be derived than already in the view instance, namely the $v_1(c,g)$ and $v_2(d,b)$. This is called query answering under *the open world assumption* [1], i.e., the available data in the views is assumed to be a subset of the data derived by the view definition programs if they are applied on the non–accessible relations e_1, e_2, e_3, e_4.

We investigate the following problem: Given a datalog program $\mathcal{P}$ and materialized views $v_1, v_2, \ldots, v_n$, construct a datalog program $\mathcal{P}_v$ with the following properties:

(i) $\mathcal{P}_v$ is a retrievable program contained in $\mathcal{P}$

(ii) Every datalog program that satisfies condition (i) is contained in $\mathcal{P}_v$.

When both conditions (i) and (ii) hold, we say that $\mathcal{P}_v$ is a *retrievable program maximally contained* in $\mathcal{P}$.

A few technical definitions: Given a view instance $\mathcal{D}_{\mathcal{V}}$, we define an *expansion* of $\mathcal{D}_{\mathcal{V}}$ to be any database (over $e_1, \ldots, e_m$) that results from $\mathcal{D}_{\mathcal{V}}$ after replacing the view facts with facts that are implied by one of the conjunctive queries produced from their definition (i.e., assuming the rules contain only EDB predicates in their bodies, we unify each tuple from $\mathcal{D}_{\mathcal{V}}$ with the head of some rule, freeze the rest of the variables in the body of the rule and replace the view fact by the facts in the body). We denote an expansion of $\mathcal{D}_{\mathcal{V}}$ by $R(\mathcal{D}_{\mathcal{V}})$). E.g., in example 1, if $\mathcal{D}_{\mathcal{V}} = \{v_1(a,b), v_2(b,c)\}$, then an expansion is $R(\mathcal{D}_{\mathcal{V}}) = \{e_1(a,1), e_2(1,b), e_3(b,c), e_4(c)\}$.

3 Retrievable disjunctive programs

Let $\mathcal{P}$ be any datalog program with output predicate q and $v_1, v_2, \ldots, v_n$ be views given by non–recursive datalog programs over the predicates of $\mathcal{P}$. We will construct in the following a disjunctive logic program $\mathcal{P}_{disj}$ and we will show that it is maximally contained in $\mathcal{P}$.

In fact, we construct a program $\mathcal{P}_{disj}$ which differs from being a datalog program in that it might contain disjunctions and function symbols in the head of some rules (which we call $\mathcal{V}^{-1}$–rules).

The construction is easy: $\mathcal{P}_{disj}$ contains: (i) all the rules of $\mathcal{P}$ except the ones that contain as subgoals EDB predicates which do not appear in any of the programs of the views and ii) For each view v_i, we complete its definition, i.e., we construct a collection of (possibly) disjunctive clauses as follows: We rewrite the definition of the view v_i as a disjunction of conjunctive queries. For this we rewrite each rule by renaming the head variables and possibly by introducing equalities in the bodies of the rules so as all the rules of the view definition have the same atom $v_i(X_1, \ldots, X_m)$ in their heads. Then, we replace all occurrences of each existential variable by a function (Skolem function) of the form $f(X_1, \ldots, X_m)$, where $X_1, \ldots, X_m$ are all variables in the head of the rule and f is a function symbol such that for each existential variable we use a different function symbol. We then take the disjunction of the bodies of the rules and rewrite it as a conjunction of disjuncts. For each disjunct, we create a new rule with this disjunct in the head and the atom $v_i(X_1, \ldots, X_m)$ in the body. Finally, all equalities are moved in the body as inequalities. The clauses obtained by applying this process to all view definitions are called $\mathcal{V}^{-1}$–rules.

Example 2. Assume that there are two materialized views v_1 and v_2 available defined as follows

$$
\begin{aligned}
&v_1(X,Z) :- e(X,Y), r(Y,Z)\\
&v_2(X,X) :- e(X,X)\\
&v_2(X,Z) :- r(X,Y), e(Y,Z)
\end{aligned}
$$

where e and r are EDB predicates.

We construct the $\mathcal{V}^{-1}$-rules as follows: By completing the definition of v_1 we take:

$$v_1(X,Z) \;\rightarrow\; \exists Y[e(X,Y) \wedge r(Y,Z)]$$

We introduce a Skolem function in order to eliminate the existential variable Y. We get

$$v_1(X,Z) \;\rightarrow\; e(X,f(X,Z)) \wedge r(f(X,Z),Z)$$

From this we obtain the following clauses

$$\begin{array}{l} e(X,f(X,Z)) :- v_1(X,Z) \\ r(f(X,Z),Z) :- v_1(X,Z) \end{array}$$

Now we complete the definition of v_2

$$v_2(X,Z) \;\rightarrow\; [Z = X \wedge e(X,X)] \vee \exists Y[r(X,Y) \wedge e(Y,Z)]$$

and introduce a Skolem function in order to eliminate the existential variable Y. We get

$$v_2(X,Z) \;\rightarrow\; [Z = X \wedge e(X,X)] \vee [r(X,g(X,Z)) \wedge e(g(X,Z),Z)]$$

transforming the right hand side into conjunctive normal form, we finaly get

$$\begin{array}{l} Z = X \vee r(X,g(X,Z)) :- v_2(X,Z) \\ Z = X \vee e(g(X,Z),Z) :- v_2(X,Z) \\ e(X,X) \vee r(X,g(X,Z)) :- v_2(X,Z) \\ e(X,X) \vee e(g(X,Z),Z) :- v_2(X,Z) \end{array}$$

Moving equalities to the right hand side, we finally get the following $\mathcal{V}^{-1}$-rules:

$$\begin{array}{l} r(X,g(X,Z)) :- v_2(X,Z), Z \neq X \\ e(g(X,Z),Z) :- v_2(X,Z), Z \neq X \\ e(X,X) \vee r(X,g(X,Z)) :- v_2(X,Z) \\ e(X,X) \vee e(g(X,Z),Z) :- v_2(X,Z) \end{array}$$

We view $\mathcal{P}_{disj}$ as a program over the EDB predicates $v_1, v_2, \ldots, v_n$. The computation of program $\mathcal{P}_{disj}$ applied on a database $\mathcal{D}$ is considered in the usual bottom up fashion only that now a) it computes disjunctions of facts [15] and b) by firing a datalog rule we mean: either we unify each subgoal with an already derived fact or we unify the subgoal with a disjunct in an already derived disjunction of facts; in the latter case the rest of the disjuncts will appear in the newly derived disjunctive fact together with the head literal and other disjuncts resulting possibly from other subgoals. (Observe that $\mathcal{V}^{-1}$–rules will be used only in the first step of the bottom up evaluation.) Derivation trees for this bottom up evaluation are defined in the usual way only that the labels on the nodes are disjunctions of facts. In the following, we will refer to a derivation tree with nodes labeled by either facts or disjunctions of facts as a *disjunctive derivation tree*.

We will refer to the *output database* of program $\mathcal{P}_{disj}$ applied on an input database, meaning the database that contains only the atomic facts without function symbols computed by $\mathcal{P}_{disj}$.

We say that program $\mathcal{P}_{disj}$ is *contained* in (*equivalent* to, respectively) a datalog program $\mathcal{P}$ iff for any input database, the output database computed by $\mathcal{P}_{disj}$ is contained in (is equal to, respectively) the output database computed by $\mathcal{P}$. If every disjunctive datalog program contained in $\mathcal{P}$ is also contained in $\mathcal{P}_{disj}$, we say that $\mathcal{P}_{disj}$ is a *retrievable* disjunctive datalog program *maximally contained* in $\mathcal{P}$.

Theorem 1. *Let $\mathcal{P}$ be any datalog program and $v_1, \ldots, v_n$ be views given by non-recursive datalog programs over the EDB predicates of $\mathcal{P}$; let $\mathcal{P}_{disj}$ be the disjunctive program constructed from $\mathcal{P}$ and the views. Then, program $\mathcal{P}_{disj}$ is a retrievable disjunctive datalog ($\neq$) program maximally contained in $\mathcal{P}$.*

Proof. (Sketch) The program $\mathcal{P}_{disj}$ contains (i) rules of $\mathcal{P}$ without view subgoals and (ii) non-recursive $\mathcal{V}^{-1}$–rules. Therefore, it can be equivalently thought of as program $\mathcal{P}$ applied on databases having disjunctions as facts. (i.e., those facts that are derived by $\mathcal{V}^{-1}$–rules).

One direction of the equivalence is easy: $\mathcal{P}_{disj}$ uses resolution in the bottom up evaluation we described, therefore it computes all the logical consequences of $\mathcal{P}$ applied on a particular disjunctive database, namely the one containing the disjunctive facts over EDB predicates of $\mathcal{P}$ implied by the view facts and their definitions in terms of the EDB predicates of $\mathcal{P}$. Hence $\mathcal{P}_{disj}$ is contained in any retrievable program maximally contained in $\mathcal{P}$.

For the other direction, we argue as follows: Let $\mathcal{P}_v$ be a maximally contained retrievable datalog program. Consider an arbitrary database $\mathcal{D}$ over the EDB predicates $v_1, \ldots, v_n$ of $\mathcal{P}$. Let $q(\mathbf{a})$ be a fact computed by $\mathcal{P}_v$. Consider all possible expansions of $\mathcal{D}$ derived by replacing a view fact by a collection of facts derived from one of the conjunctions which define this view. Because $\mathcal{P}_v$ is contained in $\mathcal{P}$, for any expansion database, fact $q(\mathbf{a})$ is computed by $\mathcal{P}$; therefore, for each expansion database, there is at least one derivation tree which derives $q(\mathbf{a})$. Let $\mathcal{T}$ be an arbitrary collection containing at least one such derivation tree for each expansion database. It suffices to prove that there is a finite derivation tree of $\mathcal{P}_{disj}$ that computes $q(\mathbf{a})$ in $\mathcal{D}$.

We conveniently define the *merging* of two (disjunctive) derivation trees, T_1 and T_2: We choose a leaf of T_2. We identify this leaf with the root of T_1 and label this node by the disjunction of the two labels. The rest of the nodes are left as are (in fact, we hang tree T_1 from T_2) only that some of the labels are changed: The label in the root of T_1 appears in the disjunction in every node of the path from the particular leaf to the root of T_2. The label of the leaf of T_1 appears in the disjunction in every node on a number of paths (arbitrary chosen) each leading from the root to a leaf of T_1.

The rest of the proof involves a combinatorial argument to show the following lemma:

Lemma 3. *There is a collection $\mathcal{T}$ of derivation trees as defined above, such that the following holds: Considering as initialization set the set of derivation trees $\mathcal{T}$, there is a sequence of mergings that produces a disjunctive derivation tree of program $\mathcal{P}_{disj}$ which computes $q(\mathbf{a})$ in $\mathcal{D}$.*

□

4 Queries and views defined by non-recursive programs

In this section we consider the case of non-recursive query datalog programs and views defined by datalog programs without recursion as well.

Theorem 2. *Suppose that the program defining the query is a non-recursive connected datalog program and the programs defining the views are all non-recursive datalog programs. Suppose that there exists a retrievable datalog program maximally contained in the query, which has no persistencies. Then there exists a retrievable datalog program maximally contained in the query which is non-recursive.*

Proof. (Sketch): Let $\mathcal{P}$ be the program defining the query and suppose there is a retrievable datalog program maximally contained in the query, call it $\mathcal{P}_v$. Suppose $\mathcal{P}_v$ is recursive. Consider any canonical database of $\mathcal{P}_v$ and suppose $\mathcal{D}$ is an expansion of this canonical database. Consider the conjunctive query generated by $\mathcal{P}$, say Q, such as there is a containment mapping from Q to $\mathcal{D}$ ($\mathcal{D}$ being viewed as a conjunctive query). Since $\mathcal{P}_v$ has no persistencies, any canonical database, and hence $\mathcal{D}$ as well, is of low ($<$ a function of the size of the program) degree. Moreover Q is connected; consequently, the image of Q in $\mathcal{D}$ may involve only a limited number ($<$ a function of the sizes of the programs) of constants of the domain of $\mathcal{D}$. Therefore $\mathcal{P}_v$ is contained in a retrievable program which itself is contained in the query, a contradiction.

We, next, present a procedure which produces a retrievable program, whenever the instance of the problem obeys certain conditions. We show that if the procedure reaches a datalog program then this is a retrievable program maximally contained in the query.

Our procedure proceeds in two steps: a) From $\mathcal{P}_{disj}$ we try to obtain a Horn program $\mathcal{P}_{Horn}$ which might contain function symbols and b) In the case we find a Horn program in the previous step, we eliminate from $\mathcal{P}_{Horn}$ the function symbols deriving the final program $\mathcal{P}_v$. The elimination of function symbols is done in a bottom up fashion as in [7, 12]. Note that in step (a) we might not obtain a Horn program.

In the first step, we try to obtain program $\mathcal{P}_{Horn}$ by applying program transformation rules to $\mathcal{P}_{disj}$. The basic transformation rule that we use is *unfolding* [10]. Unfolding in disjunctive logic programs is an extension of the unfolding in Horn clause programs. In general, the application of the unfolding rule consists of a sequence of elementary unfolding steps. Suppose that we have to unfold (elementary unfolding) a clause C at a body atom B using a clause D at a head atom B'. Suppose that θ is the most general unifier of B and B'. Then we get a clause whose body is the body of C after replacing the atom B by the body of D and whose head is the disjunction of the head of C with the head atoms of D except B'. In this clause we apply the unifier θ. Now in order to unfold a clause R at a chosen body atom B, we have to use all program clauses, which have a head atom unifiable with B. If we have more than one such head atoms we use the clause in all possible ways. Moreover, in the later case we have also to unfold the initial clause using the clauses obtained by the previous elementary unfolding steps if they also have head atoms unifiable with B and so on. The set of clauses obtained in this way may replace clause R in the program. For more details about the unfolding operation, see [10].

Besides unfolding, some clause deletion rules [9] are used, which preserve the semantics of a disjunctive logic program, (thus, we discard useless clauses). More specifically, a) we can delete a clause which is a variant of another program clause, b) we can delete a clause which is a tautology (i.e. there is an atom which appears in both the head and the body of the clause), c) we can delete a failing clause (i.e. a clause which has an atom in its body which does not unify with any head atom of the program clauses), d) we can delete a clauses which is subsumed by another program clause. We say that a clause C subsumes another clause D if there exists a substitution θ such that $head(C\theta) \subseteq head(D)$ and $body(C\theta) \subseteq body(D)$. All these deletion rules preserve the equivalence of the disjunctive programs [9].

In the following, we will also use *factoring* [15]. If two head atoms of a disjunctive clause C have a most general unifier θ then $C\theta$ is a *factor* of C. When we say that we take the *factoring closure* of a program we mean that we add to the program all factors obtained by applying factoring to all program clauses in all possible ways.

All the transformation rules presented so far preserve the equivalence of programs. Now we present, through the following two lemmas, two deletion rules, which although they do not preserve the equivalence of programs, they preserve the set of the atoms of the query predicate which are logical consequences of the programs.

Lemma 4. *Let $\mathcal{P}$ be a disjunctive logic program, p be a predicate in $\mathcal{P}$ and C be a clause in $\mathcal{P}$ of the form*

$$A_1 \vee \ldots \vee A_m :- B_1, \ldots, B_n$$

Suppose that there is an atom A_j in $\{A_1, \ldots, A_m\}$ whose predicate is different from p, such that there is no clause in $\mathcal{P}$ with a body atom which unifies with A_j. Let $\mathcal{P}' = \mathcal{P} - \{C\}$. Then, for any database $\mathcal{D}$, $\mathcal{P'}_p(\mathcal{D}) = \mathcal{P}_p(\mathcal{D})$.

Lemma 5. *Let $\mathcal{P}$ be a disjunctive logic program, p be a predicate in $\mathcal{P}$ and C be a clause in $\mathcal{P}$ of the form*

$$A_1 \vee \ldots \vee A_m :- B_1, \ldots, B_n$$

Suppose that there is no clause in $\mathcal{P}$ with a body atom whose predicate is p. Suppose also that $\mathcal{P}$ is closed under factoring and that there are two or more atoms in the head of C whose predicate is p. Let $\mathcal{P}' = \mathcal{P} - \{C\}$. Then, for any database $\mathcal{D}$, $\mathcal{P}'_p(\mathcal{D}) = \mathcal{P}_p(\mathcal{D})$.

In the following, we present a procedure which eliminates the EDB atoms from the bodies of the clauses in $\mathcal{P}$. We suppose, without loss of generality, that all predicates appearing in the body of rules of the query program are EDB predicates (as the program is non-recursive, we can always obtain a program of this form by applying unfolding to all IDB body atoms).

```
Procedure:
    Input: $\mathcal{V}^{-1}$, $\mathcal{P}$.
    Output: $\mathcal{P}_{der}$.
begin
    - Let $P = \mathcal{P} \cup \mathcal{V}^{-1}$.
    - Let $S_p$ be the set of EDB predicates in $\mathcal{P}$.
    while $S_p \neq \{\ \}$ and check2 is true do
        begin
          - Select a predicate $e$ from $S_p$.
          while check1 is true and check2 is true do
            begin
              - Select a clause $D$ from $P$ whose body contains an atom $B$
                whose predicate is $e$.
              - Unfold $D$ at $B$ by $P$. Let $P' = (P - \{D\}) \cup unfold(D, B)$.
              - Let $P''$ be the obtained from $P'$ by geting
                the factoring closure of the clauses in $unfold(D, B)$.
              - Let $P'''$ be the program obtained from $P''$ by
                applying the deletion rules in all possible ways.
              - Let $P = P'''$.
            end
          Let $S_p = S_p - \{e\}$.
        end
end.
```

The condition *check1* is *true* if: *There is a clause in P with an atom whose predicate is 'e' in its body.*

The condition *check2* is *true* if: *There is no clause in P with an EDB atom E_1 in its body which unifies with an EDB atom E_2 in the head of the same clause.*

Example 3. Consider the following datalog program $\mathcal{P}$

$$\begin{array}{ll} (1) & p(X) :- e(X,Y), r(Y) \\ (2) & p(X) :- e(Y,X), s(X) \\ (3) & p(X) :- e(X,X) \end{array}$$

and assume that there are two materialized views v_1 and v_2 available

(4) $v_1(X,Y) :- r(X), s(Y)$
(5) $v_2(X,Y) :- e(X,Y)$
(6) $v_2(X,Y) :- e(Y,X)$

Applying the procedure described in the previous section, we get $\mathcal{P}_{disj}$ which contains the rules (1)-(3) and the following $\mathcal{V}^{-1}$-rules:

(7) $r(X) :- v_1(X,Y)$
(8) $s(Y) :- v_1(X,Y)$
(9) $e(X,Y) \vee e(Y,X) :- v_2(X,Y)$

Now we apply the procedure described above in order to obtain a datalog program which has the same atomic results as P_{disj}.

We unfold (1) at '$e(X,Y)$' using (9). We get a new equivalent program, which contains the clauses $P_1 = \{2,3,7,8,9,10,11,12\}$, where:

(10) $p(X) \vee e(Y,X) :- v_2(X,Y), r(Y)$
(11) $p(X) \vee e(Y,X) :- v_2(Y,X), r(Y)$
(12) $p(X) \vee p(Y) :- v_2(Y,X), r(X), r(Y)$

Unfolding (2) at '$e(X,Y)$' using (9), (10), (11) we get $P_2 = \{3,7,8,9,10,11,12,13,14,15,16,17\}$, where:

(13) $p(X) \vee e(X,Y) :- v_2(Y,X), s(X)$
(14) $p(X) \vee e(X,Y) :- v_2(X,Y), s(X)$
(15) $p(X) \vee p(Y) :- v_2(Y,X), s(X), s(X)$
(16) $p(X) :- v_2(X,Y), r(Y), s(X)$
(17) $p(X) :- v_2(Y,X), r(Y), s(X)$

Finally, unfolding (3) at '$e(X,X)$' using (9), (10), (11), (13), (14) we get $P_3 = \{7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24\}$, where:

(18) $p(X) \vee e(X,X) :- v_2(X,X)$
(19) $p(X) \vee e(X,X) :- v_2(X,X)$
(20) $p(X) :- v_2(X,X)$
(21) $p(X) :- v_2(X,X), r(X)$
(22) $p(X) :- v_2(X,X), r(X)$
(23) $p(X) :- v_2(X,X), s(X)$
(24) $p(X) :- v_2(X,X), s(X)$

Taking the factoring closure of the program, we get from clauses (12) and (15) the clauses:

(25) $p(X) :- v_2(X,X), r(X), r(X)$
(26) $p(X) :- v_2(X,X), s(X), s(X)$

which are also added to the program. Clauses (19), (22), (24), (25) can be deleted since they are variants of other program clauses. Clauses (18), (21), (23) and (26) can be deleted since they are subsumed by clause (20). Clauses (9), (10), (11), (13) and (14) can be deleted because of lemma 4 (i.e. the atom with the predicate 'e' does not occur in any program clause). Finally, clauses (12) and (15) can be deleted from the program because of lemma 5. Therefore, the program obtained so far is $\{7, 8, 16, 17, 20\}$. Now, unfolding the EDB atoms in the bodies of (16) and (17) and deleting the redundant clauses we finally get:

$$\begin{array}{l} p(X) : -v_2(X, X) \\ p(Y) : -v_2(X, Y), v_1(Z_1, Y), v_1(X, Z_2) \\ p(X) : -v_2(X, Y), v_1(Z_1, X), v_1(Y, Z_2) \end{array}$$

This program is a retrievable Datalog program maximally contained in $\mathcal{P}$.

Concerning the procedure described above we can prove the following theorem.

Theorem 3. *The following hold for the procedure above:*

1. *The procedure always terminates.*
2. *Suppose that by applying the procedure above to a program P_{disj} we get a program P_{der} whose clause bodies do not contain any EDB atom. Then P_{der} is a datalog($\neq$) program which is maximally contained in P.*

In the following we give a syntactic sufficient condition for the algorithm to end up with datalog($\neq$) program.

Sufficient condition: *Consider the $\mathcal{V}^{-1}$-rules and the query progam $\mathcal{P}$. For each rule in $\mathcal{P}$ there is at most one atom in its body whose predicate occurs in the head of a (non-trivial) disjunctive $\mathcal{V}^{-1}$–rule.*

Theorem 4. *If the sufficient condition holds for a a set of $\mathcal{V}^{-1}$-rules and a query program $\mathcal{P}$ then by applying the procedure to $\mathcal{V}^{-1} \cup \mathcal{P}$ we get a retrievable maximally contained datalog($\neq$) program.*

5 Recursive Views

In the case where the views are given by recursive datalog programs and the query by a non-recursive datalog program $\mathcal{P}$, the problem of computing a retrievable program maximally contained in $\mathcal{P}$ is reduced to that of non-recursive view definitions (following lemma).

Lemma 6. *Let $\mathcal{P}$ be any non-recursive datalog program and $v_1, \ldots, v_n$ be views given by any datalog programs over the EDB predicates of $\mathcal{P}$. Then, there exists a positive integer N depending only on the sizes of the programs so that the following holds:*

Consider a new set of views $v'_1, \ldots v'_m$ *with view definitions* $\mathcal{P}_{v'_1}, \mathcal{P}_{v'_2}, \ldots, \mathcal{P}_{v'_m}$ *respectively: where* $\mathcal{P}_{v'_i}$ *is the disjunction of all conjunctive queries that are produced by* $\mathcal{P}_{v_i}$ *and have size less that* N*.*

If $\mathcal{P}_v$ *is a retrievable program contained in* $\mathcal{P}$ *under the new view definitions, then* $\mathcal{P}_v$ *is a retrievable program contained in* $\mathcal{P}$ *under the original view definitions.*

Proof. Let c_v be a conjunctive query generated by $\mathcal{P}_v$. Let c_v^{expi} be all expansions of c_v that result after replacing each occurrence of a view atom by one of their new definitions. Then, for each c_v^{expi}, there is a conjunctive query c_i generated by $\mathcal{P}$ and there exists a containment mapping, h_i, from c_i to c_v^{expi}.

Now, obtain all expansions oc_v^{expi}, $i = 1, 2, \ldots$, of c_v considering the original view definitions. Suppose there exists an i such that there is no conjunctive query in $\mathcal{P}$ with a containment mapping on oc_v^{expi}. Then, a view occurrence in c_v has been replaced by a long ($> N = O(2^{s \log s})$, where s is the maximum size of the programs defining the query and the views)??? conjunctive query $R(v)$; $R(v)$ appears as a subquery in oc_v^{expi}. From oc_v^{expi}, we construct a shorter expansion oc_v^{expj}: We can pump $R(v)$ to get a shorter view expansion which, though, preserves all the neighborhoods of size $\leq m$ =the size of the maximum conjunctive query in $\mathcal{P}$ (see lemmata 1, 2). We apply this pumping on all expansions that are longer than N, getting, in the end, an expansion, c_v^{expi}, of c_v under the new view definitions. There exists, though, a conjunctive query of $\mathcal{P}$ with a containment mapping on c_v^{expi}. Since oc_v^{expi} and c_v^{expi} have the same neighborhoods of size up to m, there is a containment mapping also on oc_v^{expi}; this is a contradiction. □

Theorem 5. *Let* $\mathcal{P}$ *be any non-recursive connected datalog program and* $v_1, \ldots, v_n$ *be views given by any datalog programs over the EDB predicates of* $\mathcal{P}$*. Suppose there exists a retrievable datalog program maximally contained in* $\mathcal{P}$ *which has no persistencies. Then, there exists a retrievable datalog program maximally contained in* $\mathcal{P}$ *which is non-recursive.*

Proof. An immediate consequence of the preceding lemma, and theorem 2. □

6 Chain programs

A *chain rule* is a rule over only binary predicates; moreover the first variable in the head is identical to the first variable of the first predicate in the body, the second variable in the head is identical to the last variable of the last predicate in the body and the second variable of the i-th predicate is identical to the first variable of the $i+1$-th predicate in the body. A *chain program* contains only chain rules. It is easy to see that any conjunctive query generated by a chain program $\mathcal{P}$ corresponds to a canonical database which is a simple path spelling a word over the alphabet of the EDB predicate symbols. Thus, we can view a chain program as generating a language over this alphabet; we call this language $L_{\mathcal{P}}$. Observe that for chain programs, $L_{\mathcal{P}}$ is a context free language.

In the case, though, where both the query and the views are defined by recursive programs, it becomes undecidable to answer the question whether there is a non-empty datalog program that is contained in $\mathcal{P}$ and uses only the given views as EDB predicates; it remains undecidable even in the case where both program $\mathcal{P}$ and the views are given by chain programs.

The reduction, in the following theorem is done from the containment problem of context free grammars [17, 11]

Theorem 6. *Given a datalog chain program $\mathcal{P}$ and views $v_1, v_2, \ldots, v_m$ which are also given by chain programs over the EDB predicates of $\mathcal{P}$, it is undecidable whether there is a non-empty retrievable datalog program $\mathcal{P}_v$ contained in $\mathcal{P}$.*

In some cases, though, we can have a negative answer in the question whether there exists a “simple” non–empty retrievable program, as it is stated in the theorem that follows.

Here on, we consider the following situation: A datalog query given by chain program $\mathcal{P}$ and materialized views, $v_1, v_2, \ldots, v_m$, over the EDB predicates of $\mathcal{P}$, which are also given by chain programs. Let $\mathcal{P}_v$ be a datalog program (not necessarily chain) contained in $\mathcal{P}$ that uses only $v_1, v_2, \ldots, v_m$ as EDB predicates.

Consider a datalog program $\mathcal{P}$ where both IDB and EDB predicates are binary. Take any conjunctive query c produced by $\mathcal{P}$ and take the canonical database of c. We call $\mathcal{P}$ *simple* if, for all c, the canonical database (viewed as a directed graph) contains a number of pairwise node-disjoint simple paths with endpoints on the two head constants, i.e., besides the head constants every constant (viewed as a node) has in–degree one and out–degree one. Chain programs are simple.

We need some technical concepts here on pumping lemmas for formal languages. Let w be a word over the alphabet Σ. For fixed positive integer N, we say that $w_i = ux^i vy^i w, i = 1, 2, \ldots$ is a *pumping sequence* for w if $w = uxvyw$ and $\mid xy \mid < N$. For context free languages, there exists a positive integer N, such that, for any word in a language, L, longer than N, there exists a pumping sequence $w_i, i = 1, \ldots$, such that $w_i, i = 1, \ldots$ also belongs in the language. We call such a pumping sequence a *proper* pumping sequence *wrto* language L.

Theorem 7. *Let $\mathcal{P}$ be a chain program and $v_1, v_2, \ldots, v_m$ views over the EDB predicates of $\mathcal{P}$, which are also given by chain programs $V_1, V_2, \ldots, V_m$, respectively. Suppose there exists a simple retrievable datalog program contained in $\mathcal{P}$. Then there exists a fixed positive integer N (depending only on the sizes of the programs for the views and $\mathcal{P}$), and there is a view definition V_i such that for any word w in L_{V_i}, with $\mid w \mid > N$, the following happens: There is a word $w_{\mathcal{P}}$ in $L_{\mathcal{P}}$ such that $w_{\mathcal{P}} = ww'$ and for any proper pumping sequence, $w_1, w_2, \ldots$, of w wrto L_{V_i}, there exists an infinite subsequence $w_{i_1}, w_{i_1+1}, \ldots$ such that $w_{i_1+j} w'$ also belongs to $L_{\mathcal{P}}$, for all $j = 1, 2, \ldots$.*

For an application, consider the following program $\mathcal{P}$:

$$p(X, Y) :- a(X, Z_1), p(Z_1, Z_2), p(Z_2, Z_3), a(Z_3, Y)$$
$$p(X, Y) :- b(X, Y)$$

and the views:

$$\begin{array}{l} v_1(X,Y) :- a(X,Z_1), v_1(Z_1,Z_2), a(Z_2,Y) \\ v_1(X,Y) :- b(X,Z_1), b(Z_1,Y) \\ v_2(X,Y) :- a(X,Z_1), v_2(Z_1,Y) \\ v_2(X,Y) :- a(X,Y) \end{array}$$

Theorem 8. *Given the views v_1, v_2, there is no retrievable simple datalog program contained in $\mathcal{P}$.*

Proof. Consider any word $w = w'w''$ of $L_{\mathcal{P}}$ where w' is a sufficiently large word of L_{v_1} (L_{v_2} respectively). Consider a proper pumping sequence of w' wrto L_{v_1} (L_{v_2} respectively), $w_1, w_2, \ldots$. Suppose there exists a retrievable simple program. Then, for infinitely many i's, $w_i w'$ is also a word in $L_{\mathcal{P}}$, according to the theorem above. Observe, though, that any word in $L_{\mathcal{P}}$ longer than four contains $2k+2$ as and $k+2$ bs. $w_i w'$ will not retain this balance, therefore it is not a word of $L_{\mathcal{P}}$. □

7 Conclusion

We investigated the problem of answering queries using materiliazed views. In particular, we searched for a retrievable program maximally contained in the query. In the case the query is defined by a non-recursive datalog program and the views by recursive datalog programs, we reduced the problem to that of non-recursive definitions for both the query and the views. We showed that, in the case where both the query and the views are defined by recursive datalog programs, then the problem becomes easily undecidable; we showed, though, some methods to produce negative results. It would be interesting to further pursue this latter line of research. In the case both the query and the views are defined by non-recursive datalog programs, we showed how, in certain cases, we can produce a retreivable non-recursive?? datalog program program maximally contained in the query. It seems that, further investigation towards this direction will produce interesting results. We are currently working on this.

Acknowledgment: We thank Vassilis Vassalos for helpful discussions.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. ACM Symposium on Principles of Database Systems*, 1998.

[2] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the Sixth International Conference on Database Theory*, pages 1–18. Springer-Verlag, 1997.

[3] F. Afrati, S. Cosmadakis, and M. Yannakakis. On datalog vs. polynomial time. *J. Computer and Systems Sciences*, 51(2):117–196, 1995.

[4] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseak Shim. Optimizing queries with materialized views. In *Proceedings of the 11th International Conference on Data Engineering, Los Alamitos, CA*, pages 190–200. IEEE Comput. Soc. Press, 1995.

[5] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. In *Proceedings of the Sixth International Conference on Database Theory*, pages 56–70. Springer-Verlag, 1997.

[6] Oliver M. Duschka. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, December 1997.

[7] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. 16th ACM SIGACT-SIGMOD-GIGART Symposium on Principles of Database Systems*, pages 109–116, 1997.

[8] Oliver M. Duschka and Michael R. Genesereth. Query planning with disjunctive sources. In *Proc. of the AAAI-98 Workshop on AI and Information Integration*, 1998.

[9] M. Gergatsoulis. Correctness-preserving transformations for disjunctive logic programs. Demo 97/2, Institute of Informatics & Telecom. NCSR 'Demokritos', 1997.

[10] Manolis Gergatsoulis. Unfold/fold transformations for disjunctive logic programs. *Information Processing Letters*, 62(1):23–29, April 1997.

[11] M. A. Harrison. *Introduction to formal language theory.* Addison-Wesley, 1978.

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

[13] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, 1995.

[14] Alon Y. Levy, Divest Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 1995. Special Issue on Networked Information Discovery and Retrieval.

[15] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.

[16] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. 14th ACM SIGACT-SIGMOD-GIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[17] Oded Shmueli. Decidability and expressiveness aspects of logic queries. In *Proc. 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 237–249, 1987.

[18] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I & II. Computer Science Press, 1989.

[19] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory*, pages 19–40. Springer-Verlag, 1997.

[20] Jeffrey D. Ullman and Allen Van Gelder. Parallel complexity of logical query programs. In *Proc. 27th IEEE Symp. on Foundations of Comp. Sci.*, pages 438–454, 1986.

Answering Queries Using Materialized Views with Disjunctions[⋆] [⋆⋆]

Foto N. Afrati[1], Manolis Gergatsoulis[2], and Theodoros Kavalieros[1]

[1] Dept. of Electrical and Computer Engineering,
National Technical University of Athens,
157 73 Athens, Greece,
{afrati,kavalier}@softlab.ece.ntua.gr
[2] Inst. of Informatics & Telecom.,
N.C.S.R. 'Demokritos',
153 10 A. Paraskevi Attikis, Greece
manolis@iit.demokritos.gr

Abstract. We consider the problem of answering datalog queries using materialized views. More specifically, queries are rewritten to refer to views instead of the base relations over which the queries were originally written. Much work has been done on program rewriting that produces an equivalent query. In the context of information integration, though, the importance of using views to infer as many answers as possible has been pointed out. Formally, the problem is: Given a datalog program $\mathcal{P}$ is there a datalog program $\mathcal{P}_v$ which uses only views as EDB predicates and (i) produces a subset of the answers that $\mathcal{P}$ produces and (ii) any other program $\mathcal{P}'_v$ over the views with property (i) is contained in $\mathcal{P}_v$? In this paper we investigate the problem in the case of disjunctive view definitions.

1 Introduction

A considerable amount of recent work has focused on using materialized views to answer queries [1, 7, 5, 12, 16, 13, 4]. This issue may arise in several situations, e.g., if the relations mentioned in the query are not actually stored or are impossible to consult or are very costly to access. The ability to use views is important in many applications, including information integration, where information sources are considered to store materialized views over a global database schema [2, 19, 14].

[⋆] This work has been partially supported by the Greek General Secretariat of Research and Technology under the project "Logic Programming Systems and Environments for developing Logic Programs" of ΠENEΔ'95, contract no 952.

[⋆⋆] This paper appears in Proc. of the 7th International Conference: Database Theory (ICDT'99), C. Beeri and P. Buneman (editors), LNCS 1540, p.p. 435-452, Springer-Verlag 1999.

Suppose that we want to integrate three databases that provide flight information. These three databases can be seen as views over the EDB predicates $international\_flight(X,Y)$ and $local\_flight(X,Y)$ (which have the meaning that there is a non-stop international flight from a greek city X to city Y and, there is a non-stop local flight from a greek city X to a greek city Y respectively):

$$\begin{aligned}&v_1(X) :- \ international\_flight(Athens, X)\\&v_1(X) :- \ international\_flight(Rhodes, X)\\&v_2(X,Y) :- \ local\_flight(X, Athens), local\_flight(Athens, Y)\\&v_3(X) :- \ local\_flight(X, Rhodes)\end{aligned}$$

Suppose that the user is interested in whether there is a way to fly from a greek city X to a city Y in another country making at most one intermediate connection in a greek city and then a direct international flight i.e. the following query $\mathcal{P}$, is asked:

$$\begin{aligned}&p(X,Y) :- \ international\_flight(X,Y)\\&p(X,Y) :- \ local\_flight(X,Z), international\_flight(Z,Y)\end{aligned}$$

The user does not have access to the databases that contain the facts in the predicates *international_flight* and *local_flight* but only to the views. Using only the materialized views v_1, v_2 and v_3, it is not possible to find all the answers. The best one can do is to retrieve all the flights that use either the international airport of *Athens* or the international airport of *Rhodes*, i.e. ask the following query $\mathcal{P}_v$, on the available data:

$$\begin{aligned}&p(Athens,Y) :- \ v_2(W, Rhodes), v_1(Y)\\&p(Rhodes,Y) :- \ v_2(Rhodes, W), v_1(Y)\\&p(Athens,Y) :- \ v_3(Athens), v_1(Y)\\&p(X,Y) :- \ v_3(X), v_2(X,W), v_1(Y)\end{aligned}$$

In fact, the program $\mathcal{P}_v$ which is said to be a *retrievable program* (i.e., it contains only the view predicates as EDB predicates), is maximal in the sense that any datalog program which uses only the views v_1, v_2, v_3 as EDB predicates and produces only *correct* answers is contained in $\mathcal{P}$. Such a program is said to be a *retrievable maximally contained program.*

The problem that we consider here is : Given a datalog query and views defined over its EDB predicates, is there a retrievable maximally contained datalog program?

Previous work on this problem is done in [7, 12] where they restricted views to being defined by conjunctive queries, whereas in [8, 6] disjunctive view definitions are considered and is shown how to express a retrievable maximally contained program in disjunctive datalog with inequality. In [1], the problem of computing correct (called *certain*) answers is considered, also, in the general case where views are defined by datalog programs; they investigate the data complexity problem under both the closed world assumption and the open world assumption.

In this paper, we investigate the case where views have disjunctions in their definition. We prove the following results:

a) In the case where both the query and the views are given by non-recursive datalog programs, we identify a non-trivial class of instances of the problem where there exists a retrievable datalog($\neq$) program maximally contained in the query; in this case, we give an algorithm to obtain it; this program computes *all* correct answers. We know, though, that, in general such a program does not exist [1].

b) When the views are defined by recursive datalog programs and the query by non-recursive datalog program, we reduce the problem to that of non-recursive view definitions.

c) In the case the query is given by a recursive datalog program and the views by non-recursive datalog programs, we construct a simple disjunctive logic program, which, applied on a view instance, computes all correct answers whenever it halts. This result is also obtained in [8] under a similar technique. We also show how, in certain cases, this disjunctive program can be transformed into a datalog($\neq$) program thus obtaining the results mentioned in (a) above.

d) Finally, when both the views and the query are given by recursive programs, we prove that it is undecidable whether there exists a non-empty retrievable program $\mathcal{P}_v$. Still, we give a criterion that gives a negative answer, in some special cases. In fact, the undecidability result holds even in the case of simple chain programs [20].

Other related work is done in [5, 16, 13, 4], where the approaches used essentially look for rewritings producing equivalent queries. It is known [13] that the problem of finding a rewriting that produces an equivalent program is NP-complete, in the case, where both the query and the views are defined by disjunctions of conjunctive queries. Other work concern conjunctive queries (the views and the question asked) and include [13, 16] where they give nondeterministic polynomial time algorithms to find either a single retrievable query equivalent to the given query or a set of retrievable queries whose union is contained in the given query Q and that contains any other retrievable datalog query that is contained in Q. Special classes of conjunctive queries are examined in [5].

2 Preliminaries

2.1 Logic programs and datalog

A *disjunctive clause* C is a formula of the form [15]

$$A_1 \vee \ldots \vee A_m :- B_1, \ldots, B_n$$

where $m \geq 1$, $n \geq 0$ and $A_1, \ldots, A_m, B_1, \ldots, B_n$ are atoms. If $n = 0$, C is called a *positive disjunctive clause*. If $m = 1$, C is called a *definite clause* or *Horn clause*. A *definite/disjunctive program* is a finite set of definite/disjunctive clauses. A *Datalog program* is a set of function free Horn clauses. A *Datalog($\neq$) program* is a set of function free Horn clauses whose bodies are allowed to contain atoms whose predicate is the built-in inequality predicate ($\neq$). The left hand side of a rule is called the *head* of the rule and the right hand side is the *body* of the rule.

A predicate is an *intensional database predicate*, or *IDB predicate*, in a program $\mathcal{P}$ if it appears at the head of a rule in $\mathcal{P}$, otherwise it is an *extensional database (EDB) predicate.* A *conjunctive query* is a single non-recursive function-free Horn rule.

Let D be a finite set. A *database* over domain D is a finite relational structure $\mathcal{D} = (D, r_1, \ldots, r_n)$, where each r_i is a relation over D; a *substructure* of a database is a relational structure $(D', r'_1, \ldots, r'_n)$, where each r'_i contains a subset of the facts contained in r_i.

A *query* is a function from databases (of some fixed type) to relations of fixed arity over the same domain; it has to be *generic*, i.e., invariant under renamings of the domain.

Datalog programs may be viewed as a declarative query language with the following semantics. Let $\mathcal{D}$ be a database, thought of as a collection of facts about the EDB predicates of a program $\mathcal{P}$. Let $Q^k_{\mathcal{P},p}(\mathcal{D})$ be the collection of facts about an IDB predicate p that can be deduced from $\mathcal{D}$ by at most k applications of the rules in $\mathcal{P}$. If we consider p the *goal* predicate (or *output* predicate), then $\mathcal{P}$ expresses a query $Q_{\mathcal{P},p}$, where

$$Q_{\mathcal{P},p}(\mathcal{D}) = \bigcup_{k \geq 0} Q^k_{\mathcal{P},p}(\mathcal{D})$$

We will sometimes write $Q_{\mathcal{P}}$ or $\mathcal{P}_p(\mathcal{D})$ instead of $Q_{\mathcal{P},p}$.

The above definition also gives a particular algorithm to compute $Q_{\mathcal{P},p}$: initialize the IDB predicates to be empty, and repeatedly apply the rules to add tuples to the IDB predicates, until no new tuples can be added. This is known as *bottom-up evaluation.* A *derivation tree* is a tree depicting the bottom up evaluation of a specific derived fact in the IDB. Its nodes are labeled by facts and each node with its children correspond to an instantiation of a rule in $\mathcal{P}$ [18].

Given a datalog program, we can define a *dependency graph*, whose nodes are the predicate names appearing in the rules. There is an edge from the node of predicate p to the node of predicate p' if p' appears in the body of a rule whose head predicate is p. The program is *recursive* if there is a cycle in the dependency graph. We define, also, the *adorned dependency graph* as follows: We assign labels (possible none or more than one) on each edge: Suppose there is a rule with head predicate symbol p and predicate symbol p' appears somewhere in the body; suppose that the i-th argument of p is the same variable as the j-th argument of p'. Then, on edge $e = (p, p')$ we assign a label $l = i \rightarrow j$; we call the pair (e, l) a *p-pair.* We define a *perfect cycle* in the adorned dependency graph to be a sequence of p-pairs $(e_1, l_1), \ldots, (e_k, l_k)$ such that the edges $e_1, \ldots, e_k$ form a cycle and, moreover, the following holds: for all $q = 1, \ldots, k-1$ if $l_q = i \rightarrow j$ then $l_{q+1} = j \rightarrow j'$ for some j' and, if $l_k = i \rightarrow j$ then $l_1 = j \rightarrow j'$ for some j'. We say that a datalog program *has no presistencies* iff there is no perfect cycle in the adorned dependency graph.

Given any datalog program $\mathcal{P}$, we can unwind the rules several times and produce a conjunctive query (over the EDB predicates of $\mathcal{P}$) generated by $\mathcal{P}$. Program $\mathcal{P}$ can, therefore, be viewed as being equivalent to an infinite disjunction of all these conjunctive queries. Considering a conjunctive query, we freeze the body of the query by turning each of the subgoals into facts in a database; we obtain, thus, the *canonical database* of this query. The constants in the canonical database that correspond to variables that also appear in the head of the query are conveniently called *head constants.* If all EDB predicates are binary, we may view a canonical database as a labeled graph. Correspondingly, we can refer to a canonical database *generated* by a datalog program. We say that a datalog program is *connected* iff all head variables of the program rules occur also in the bodies of the corresponding rules and any canonical database (viewed as a hypergraph) is connected.

A *containment mapping* from a conjunctive query Q_1 to a conjunctive query Q_2, is a function, h, that maps all variables of Q_1 on variables of Q_2, so that whenever an atom $p(X_1, X_2, \ldots)$ occurs in Q_1, then an atom $p(h(X_1), h(X_2), \ldots)$ occurs in Q_2.

The following are technical lemmata used in the proof of the main results.

Lemma 1. *Let $\mathcal{P}$ be a Datalog program and c a conjunctive query. There exists a positive integer N depending only on the sizes of the program and the query so that the following holds: If there is a conjunctive query c_i generated by $\mathcal{P}$ such that there is a containment mapping from c to c_i, then there is a conjunctive query c_j generated by $\mathcal{P}$, of size $< N$, such that there is a containment mapping from c to c_j.*

Proof. First, we note, that, given a sufficiently large conjunctive query generated by a datalog program, we can apply a pumping on it, obtaining a shorter conjunctive query generated by the same program (see [3]).

Suppose c_i is of size $> N$. The containment mapping from c to c_i maps c on at most $|c|$ constants in c_i. Thus, there are $> N - |c|$ constants in c_i that are not involved in this mapping. Consider this "free" part of c_i and do on it a pumping, obtaining, thus a shorter conjunctive query, on which the containment mapping is preserved. ☐

Definition 1. *Consider a database. A* neighborhood of size *k in the database is a connected substructure of size k. If we view the database as the canonical database of a conjunctive query, then, we can refer to neighborhoods in a conjunctive query accordingly.*

Lemma 2. *Suppose two conjunctive queries Q_1, Q_2 such that: each neighborhood of size up to k appears in one query iff it appears in the other. Then, for any conjunctive query Q of size up to k, there is containment mapping from Q to the query Q_1 iff there is containment mapping from Q to Q_2.*

2.2 Retrievable programs

Let $\mathcal{P}$ be any datalog program over EDB predicates $e_1, e_2, \ldots, e_m$ and with output predicate q. Let $v_1, v_2, \ldots, v_n$ be views defined by datalog programs over EDB predicates $e_1, e_2, \ldots, e_m$; call $\mathcal{P}_\mathcal{V}$ the union of programs that define the views. We want to answer the query $Q_{\mathcal{P}}$ assuming that we do not have access to database relations $e_1, e_2, \ldots, e_m$, but only to views $v_1, \ldots, v_n$; we need a "certain" answer no matter which is the database that yielded the given view instance.

Definition 2. *A* certain *answer to query $Q_{\mathcal{P}}$, given a view instance $\mathcal{D}_{\mathcal{V}}$, is a tuple t such that: for each database $\mathcal{D}$ over $e_1, e_2, \ldots, e_m$ with $\mathcal{D}_{\mathcal{V}} \subseteq \mathcal{P}_{\mathcal{V}}(\mathcal{D})$, t belongs to $Q_{\mathcal{P}}(\mathcal{D})$.*

The above definition assumes the open world assumption; under the closed world assumption, the definition requires that $\mathcal{D}_{\mathcal{V}} = \mathcal{P}_{\mathcal{V}}(\mathcal{D})$; to compute a certain answer in this case is harder (see [1]).

Let $\mathcal{P}_v$ be a datalog program over EDB predicates $v_1, v_2, \ldots, v_n$. We say that $\mathcal{P}_v$ *is a retrievable program contained* in $\mathcal{P}$ if, for each view instance $\mathcal{D}_{\mathcal{V}}$, all tuples in $\mathcal{P}_v(\mathcal{D}_{\mathcal{V}})$ are certain answers.

Example 1. Let views v_1, v_2 be defined over the EDB predicates e_1, e_2, e_3, e_4:

$$\begin{aligned}
v_1(X,Y) &:- \ e_1(X,Y) \\
v_1(X,Y) &:- \ e_1(X,Z), e_2(Z,Y) \\
v_2(X,Y) &:- \ e_3(X,Y), e_4(Y)
\end{aligned}$$

and the query $\mathcal{P}$ by:

$$p(X) :- \ e_3(Y,X), e_1(X,Z)$$

Then a retrievable program maximally contained in $\mathcal{P}_v$, is:

$$p_v(X) :- \ v_2(Y,X), v_1(X,Z)$$

If $\mathcal{P}_v$ is applied on an instance of v_1, v_2, it will produce a set of answers; this is a subset of the answers which will be produced if query $\mathcal{P}$ is applied on some instance of the relations e_1, e_2, e_3, e_4 which yields the given instance of v_1, v_2. For example, suppose $v_1 = \{(a,b),(b,c),(b,d),(c,f)\}$ and $v_2 = \{(d,a)\}$. Then $\mathcal{P}_v$ will produce the set of answers $\{(a)\}$. A database that yields the above view instance could have been either $\mathcal{D}_1 = \{e_1(a,b), e_1(b,c), e_1(c,f),\ e_2(c,d), e_3(d,a),\ e_3(b,c),\ e_4(a)\}$ or $\mathcal{D}_2 = \{e_1(a,b), e_1(b,d),\ e_1(c,f), e_2(d,c),\ e_2(f,g),\ e_3(d,a),\ e_3(d,b),\ e_4(a),\ e_4(c),\ e_4(b)\}$ (or infinitely many other alternatives). If we apply $\mathcal{P}$ on $\mathcal{D}_1$, we get the answers $\{(a),(c)\}$, whereas if we apply $\mathcal{P}$ on $\mathcal{D}_2$ we get the answers $\{(a),(b)\}$; tuple (a) is the only answer common to both. It turns out that, for any view instance, program $\mathcal{P}_v$ provably derives exactly those answers that are common to all relational structures over e_1, e_2, e_3, e_4 that yield this view instance. Note, that if the programs that define the views are applied on $\mathcal{D}_2$, then more tuples will be derived than already in the view instance, namely the $v_1(c,g)$ and $v_2(d,b)$. This is called query answering under *the open world assumption* [1], i.e., the available data in the views is assumed to be a subset of the data derived by the view definition programs if they are applied on the non–accessible relations e_1, e_2, e_3, e_4. □

We investigate the following problem: Given a datalog program $\mathcal{P}$ and materialized views $v_1, v_2, \ldots, v_n$, construct a datalog program $\mathcal{P}_v$ with the following properties:

(i) $\mathcal{P}_v$ is a retrievable program contained in $\mathcal{P}$

(ii) Every datalog program that satisfies condition (i) is contained in $\mathcal{P}_v$.

When both conditions (i) and (ii) hold, we say that $\mathcal{P}_v$ is a *retrievable program maximally contained* in $\mathcal{P}$.

A few technical definitions: Given a view instance $\mathcal{D}_{\mathcal{V}}$, we define an *expansion* of $\mathcal{D}_{\mathcal{V}}$ to be any database (over $e_1, \ldots, e_m$) that results from $\mathcal{D}_{\mathcal{V}}$ after replacing the view facts with facts that are implied by one of the conjunctive queries produced from the view definitions (i.e., assuming the rules contain only EDB predicates in their bodies, we unify each tuple from $\mathcal{D}_{\mathcal{V}}$ with the head of some rule, freeze the rest of the variables in the body of the rule and replace the view fact by the facts in the body). We denote an expansion of $\mathcal{D}_{\mathcal{V}}$ by $R(\mathcal{D}_{\mathcal{V}})$). E.g., in example 1, if $\mathcal{D}_{\mathcal{V}} = \{v_1(a,b), v_2(b,c)\}$, then an expansion is $R(\mathcal{D}_{\mathcal{V}}) = \{e_1(a,1), e_2(1,b), e_3(b,c), e_4(c)\}$.

3 Retrievable disjunctive programs

Let $\mathcal{P}$ be any datalog program with output predicate q and $v_1, v_2, \ldots, v_n$ be views given by non–recursive datalog programs over the predicates of $\mathcal{P}$. We will construct in the following a disjunctive logic program $\mathcal{P}_{disj}$ and we will show that it is maximally contained in $\mathcal{P}$.

In fact, we construct a program $\mathcal{P}_{disj}$ which differs from being a datalog program in that it might contain disjunctions and function symbols in the head of some rules (which we call $\mathcal{V}^{-1}$–rules).

The construction is easy: $\mathcal{P}_{disj}$ contains: (i) all the rules of $\mathcal{P}$ except the ones that contain as subgoals EDB predicates which do not appear in any of the programs of the views and ii) For each view v_i, we complete its definition, i.e., we construct a collection of (possibly) disjunctive clauses as follows: We rewrite the definition of the view v_i as a disjunction of conjunctive queries. For this we rewrite each rule by renaming the head variables and possibly by introducing equalities in the bodies of the rules so as all the rules of the view definition have the same atom $v_i(X_1, \ldots, X_m)$ in their heads. Then, we replace all occurrences of each existential variable by a function (Skolem function) of the form $f(X_1, \ldots, X_m)$, where $X_1, \ldots, X_m$ are all variables in the head of the rule and f is a function symbol such that for each existential variable we use a different function symbol. We then take the disjunction of the bodies of the rules and rewrite it as a conjunction of disjuncts. For each disjunct, we create a new rule with this disjunct in the head and the atom $v_i(X_1, \ldots, X_m)$ in the body. Finally, all equalities are moved in the body as inequalities. The clauses obtained by applying this process to all view definitions are called $\mathcal{V}^{-1}$–rules.

Example 2. Assume that there are two materialized views v_1 and v_2 available defined as follows

$$\begin{array}{l} v_1(X,Z) :- e(X,Y), r(Y,Z) \\ v_2(X,X) :- e(X,X) \\ v_2(X,Z) :- r(X,Y), e(Y,Z) \end{array}$$

where e and r are EDB predicates.

We construct the $\mathcal{V}^{-1}$-rules as follows: By completing the definition of v_1 we take:

$$v_1(X,Z) \ \rightarrow \ \exists Y[e(X,Y) \wedge r(Y,Z)]$$

We introduce a Skolem function in order to eliminate the existential variable Y. We get

$$v_1(X,Z) \ \rightarrow \ e(X,f(X,Z)) \wedge r(f(X,Z),Z)$$

From this we obtain the following clauses

$$\begin{array}{l} e(X,f(X,Z)) :- v_1(X,Z) \\ r(f(X,Z),Z) :- v_1(X,Z) \end{array}$$

Now we complete the definition of v_2

$$v_2(X,Z) \ \rightarrow \ [Z = X \wedge e(X,X)] \vee \exists Y[r(X,Y) \wedge e(Y,Z)]$$

and introduce a Skolem function in order to eliminate the existential variable Y. We get

$$v_2(X,Z) \ \rightarrow \ [Z = X \wedge e(X,X)] \vee [r(X,g(X,Z)) \wedge e(g(X,Z),Z)]$$

transforming the right hand side into conjunctive normal form, we finally get

$$\begin{array}{l} Z = X \vee r(X,g(X,Z)) :- v_2(X,Z) \\ Z = X \vee e(g(X,Z),Z) :- v_2(X,Z) \\ e(X,X) \vee r(X,g(X,Z)) :- v_2(X,Z) \\ e(X,X) \vee e(g(X,Z),Z) :- v_2(X,Z) \end{array}$$

Moving equalities to the right hand side, we finally get the following $\mathcal{V}^{-1}$-rules:

$$\begin{array}{l} r(X,g(X,Z)) :- v_2(X,Z), Z \neq X \\ e(g(X,Z),Z) :- v_2(X,Z), Z \neq X \\ e(X,X) \vee r(X,g(X,Z)) :- v_2(X,Z) \\ e(X,X) \vee e(g(X,Z),Z) :- v_2(X,Z) \end{array}$$

□

We view $\mathcal{P}_{disj}$ as a program over the EDB predicates $v_1, v_2, \ldots, v_n$. The computation of program $\mathcal{P}_{disj}$ applied on a database $\mathcal{D}$ is considered in the usual bottom up fashion only that now a) it computes disjunctions of facts [15] and b) by firing a datalog rule we mean: either we unify each subgoal with an already derived fact or we unify the subgoal with a disjunct in an already derived disjunction of facts; in the latter case the rest of the disjuncts will appear in the newly derived disjunctive fact together with the head literal and other disjuncts resulting possibly from other subgoals. (Observe that $\mathcal{V}^{-1}$–rules will be used only in the first step of the bottom up evaluation.) Derivation trees for this bottom up evaluation are defined in the usual way only that the labels on the nodes are disjunctions of facts. In the following, we will refer to a derivation tree with nodes labeled by either facts or disjunctions of facts as a *disjunctive derivation tree.*

We will refer to the *output database* of program $\mathcal{P}_{disj}$ applied on an input database, meaning the database that contains only the atomic facts without function symbols computed by $\mathcal{P}_{disj}$.

We say that program $\mathcal{P}_{disj}$ is *contained* in (*equivalent* to, respectively) a datalog program $\mathcal{P}$ iff for any input database, the output database computed by $\mathcal{P}_{disj}$ is contained in (is equal to, respectively) the output database computed by $\mathcal{P}$. If every disjunctive datalog program contained in $\mathcal{P}$ is also contained in $\mathcal{P}_{disj}$, we say that $\mathcal{P}_{disj}$ is a *retrievable* disjunctive datalog program *maximally contained* in $\mathcal{P}$.

Theorem 1. *Let $\mathcal{P}$ be any datalog program and $v_1, \ldots, v_n$ be views given by non-recursive datalog programs over the EDB predicates of $\mathcal{P}$; let $\mathcal{P}_{disj}$ be the disjunctive program constructed from $\mathcal{P}$ and the views. Then, program $\mathcal{P}_{disj}$ is a retrievable disjunctive datalog($\neq$) program maximally contained in $\mathcal{P}$.*

Proof. (Sketch) The program $\mathcal{P}_{disj}$ contains (i) rules of $\mathcal{P}$ without view subgoals and (ii) non-recursive $\mathcal{V}^{-1}$–rules. Therefore, it can be equivalently thought of as program $\mathcal{P}$ applied on databases having disjunctions as facts. (i.e., those facts that are derived by $\mathcal{V}^{-1}$–rules).

One direction is easy: $\mathcal{P}_{disj}$ uses resolution in the bottom up evaluation we described, therefore it computes all the logical consequences of $\mathcal{P}$ applied on a particular disjunctive database, namely the one containing the disjunctive facts over EDB predicates of $\mathcal{P}$ implied by the view facts and their definitions in terms of the EDB predicates of $\mathcal{P}$. Hence $\mathcal{P}_{disj}$ is contained in any retrievable program maximally contained in $\mathcal{P}$.

For the other direction, we argue as follows: Let $\mathcal{P}_v$ be a maximally contained retrievable datalog program. Consider an arbitrary database $\mathcal{D}$ over the EDB predicates $v_1, \ldots, v_n$ of $\mathcal{P}$. Let $q(\mathbf{a})$ be a fact computed by $\mathcal{P}_v$. Consider all possible expansions of $\mathcal{D}$ derived by replacing a view fact by a collection of facts derived from one of the conjunctions which define this view. Because $\mathcal{P}_v$ is contained in $\mathcal{P}$, for any expansion database, fact $q(\mathbf{a})$ is computed by $\mathcal{P}$; therefore, for each expansion database, there is at least one derivation tree which derives $q(\mathbf{a})$. Let $\mathcal{T}$ be an arbitrary collection containing at least one such derivation tree for each expansion database. It suffices to prove that there is a finite derivation tree of $\mathcal{P}_{disj}$ that computes $q(\mathbf{a})$ in $\mathcal{D}$.

We conveniently define the *merging* of two (disjunctive) derivation trees, T_1 and T_2: We choose a leaf of T_2. We identify this leaf with the root of T_1 and label this node by the disjunction of the two labels. The rest of the nodes are left as are (in fact, we hang tree T_1 from T_2) only that some of the labels are changed: The label in the root of T_1 appears in the disjunction in every node of the path from the particular leaf to the root of T_2. The label of the leaf of T_1 appears in the disjunction in every node on a number of paths (arbitrary chosen) each leading from the root to a leaf of T_1.

The rest of the proof involves a combinatorial argument to show the following lemma:

Lemma 3. *There is a collection $\mathcal{T}$ of derivation trees as defined above, such that the following holds: Considering as initialization set the set of derivation trees $\mathcal{T}$, there is a sequence of mergings that produces a disjunctive derivation tree of program $\mathcal{P}_{disj}$ which computes $q(\mathbf{a})$ in $\mathcal{D}$.*

□

4 Queries and views defined by non-recursive programs

In this section we consider the case of non-recursive query datalog programs and views defined by datalog programs without recursion as well.

Theorem 2. *Suppose that the program defining the query is a non-recursive connected datalog program and the programs defining the views are all non-recursive datalog programs. Suppose that there exists a retrievable datalog program maximally contained in the query, which has no persistencies. Then there exists a retrievable datalog program maximally contained in the query which is non-recursive.*

Proof. (Sketch): Let $\mathcal{P}$ be the program defining the query and suppose there is a retrievable datalog program maximally contained in the query, call it $\mathcal{P}_v$. Suppose $\mathcal{P}_v$ is recursive. Consider any canonical database of $\mathcal{P}_v$ and suppose $\mathcal{D}$ is an expansion of this canonical database. Consider the conjunctive query generated by $\mathcal{P}$, say Q, such as there is a containment mapping from Q to $\mathcal{D}$ ($\mathcal{D}$ being viewed as a conjunctive query). Since $\mathcal{P}_v$ has no persistencies, any canonical database, and hence $\mathcal{D}$ as well, is of low ($<$ a function of the size of the program) degree. Moreover Q is connected; consequently, the image of Q in $\mathcal{D}$ may involve only a limited number ($<$ a function of the sizes of the programs) of constants of the domain of $\mathcal{D}$. Therefore $\mathcal{P}_v$ is contained in a retrievable program which itself is contained in the query, a contradiction. □

We, next, present a procedure which produces a retrievable program, whenever the instance of the problem obeys certain conditions. We show that if the procedure reaches a datalog program then this is a retrievable program maximally contained in the query.

Our procedure proceeds in two steps: a) From $\mathcal{P}_{disj}$ we try to obtain a Horn program $\mathcal{P}_{Horn}$ which might contain function symbols and b) In the case we find a Horn program in the previous step, we eliminate from $\mathcal{P}_{Horn}$ the function symbols deriving the final program $\mathcal{P}_v$. The elimination of function symbols is done in a bottom up fashion as in [7, 12]. Note that in step (a) we might not obtain a Horn program.

In the first step, we try to obtain program $\mathcal{P}_{Horn}$ by applying program transformation rules to $\mathcal{P}_{disj}$. The basic transformation rule that we use is *unfolding* [10]. Unfolding in disjunctive logic programs is an extension of the unfolding in Horn clause programs. In general, the application of the unfolding rule consists of a sequence of elementary unfolding steps. Suppose that we have to unfold (elementary unfolding) a clause C at a body atom B using a clause D at a head atom B'. Suppose that θ is the most general unifier of B and B'. Then we get a clause whose body is the body of C after replacing the atom B by the body of D and whose head is the disjunction of the head of C with the head atoms of D except B'. In this clause we apply the unifier θ. Now in order to unfold a clause R at a chosen body atom B, we have to use all program clauses, which have a head atom unifiable with B. If we have more than one such head atoms we use the clause in all possible ways. Moreover, in the later case we have also to unfold the initial clause using the clauses obtained by the previous elementary unfolding steps if they also have head atoms unifiable with B and so on. The set of clauses obtained in this way, denoted by $unfold(C, B)$, may replace clause R in the program. For more details about the unfolding operation, see [10].

Besides unfolding, some clause deletion rules [9] are used, which preserve the semantics of a disjunctive logic program, (thus, we discard useless clauses). More specifically, a) we can delete a clause which is a variant of another program clause, b) we can delete a clause which is a tautology (i.e. there is an atom which appears in both the head and the body of the clause), c) we can delete a failing clause (i.e. a clause which has an atom in its body which does not unify with any head atom of the program clauses), d) we can delete a clauses which is subsumed by another program clause. We say that a clause C subsumes another clause D if there exists a substitution θ such that $head(C\theta) \subseteq head(D)$ and $body(C\theta) \subseteq body(D)$. All these deletion rules preserve the equivalence of the disjunctive programs [9].

In the following, we will also use *factoring* [15]. If two head atoms of a disjunctive clause C have a most general unifier θ then $C\theta$ is a *factor* of C. When we say that we take the *factoring closure* of a program we mean that we add to the program all factors obtained by applying factoring to all program clauses in all possible ways.

All the transformation rules presented so far preserve the equivalence of programs. Now we present, through the following two lemmas, two deletion rules, which although they do not preserve the equivalence of programs, they preserve the set of the atoms of the query predicate which are logical consequences of the programs.

Lemma 4. *Let $\mathcal{P}$ be a disjunctive logic program, p be a predicate in $\mathcal{P}$ and C be a clause in $\mathcal{P}$ of the form*

$$A_1 \vee \ldots \vee A_m :- \; B_1, \ldots, B_n$$

Suppose that there is an atom A_j *in* $\{A_1, \ldots, A_m\}$ *whose predicate is different from p, such that there is no clause in* $\mathcal{P}$ *with a body atom which unifies with* A_j. *Let* $\mathcal{P}' = \mathcal{P} - \{C\}$. *Then, for any database* $\mathcal{D}$, $\mathcal{P}'_p(\mathcal{D}) = \mathcal{P}_p(\mathcal{D})$.

Lemma 5. *Let* $\mathcal{P}$ *be a disjunctive logic program, p be a predicate in* $\mathcal{P}$ *and C be a clause in* $\mathcal{P}$ *of the form*

$$A_1 \vee \ldots \vee A_m :- B_1, \ldots, B_n$$

Suppose that there is no clause in $\mathcal{P}$ *with a body atom whose predicate is p. Suppose also that* $\mathcal{P}$ *is closed under factoring and that there are two or more atoms in the head of C whose predicate is p. Let* $\mathcal{P}' = \mathcal{P} - \{C\}$. *Then, for any database* $\mathcal{D}$, $\mathcal{P}'_p(\mathcal{D}) = \mathcal{P}_p(\mathcal{D})$.

In the following, we present an algorithm which gives a systematic way to apply the above described operations on $\mathcal{V}^{-1} \cup \mathcal{P}$ deriving, in certain cases, a retrievable datalog($\neq$) program maximally contained in $\mathcal{P}$. We suppose, without loss of generality, that all predicates appearing in the body of rules of the query program are EDB predicates (as the program is non-recursive, we can always obtain a program of this form by applying unfolding to all IDB body atoms).

```
Procedure:
    Input: $\mathcal{V}^{-1}$, $\mathcal{P}$.
    Output: $\mathcal{P}_{der}$.
begin
    - Let $P = \mathcal{P} \cup \mathcal{V}^{-1}$.
    - Let $S_p$ be the set of EDB predicates in $\mathcal{P}$.
    while $S_p \neq \{\ \}$ and check2 is true do
        begin
          - Select a predicate $e$ from $S_p$.
          while check1(e) is true and check2 is true do
            begin
              - Select a clause $D$ from $P$ whose body contains an atom $B$
                whose predicate is $e$.
              - Unfold $D$ at $B$ by $P$. Let $P' = (P - \{D\}) \cup unfold(D, B)$.
              - Let $P''$ be the obtained from $P'$ by getting
                the factoring closure of the clauses in $unfold(D, B)$.
              - Let $P'''$ be the program obtained from $P''$ by
                applying the deletion rules.
              - Let $P = P'''$.
            end
          Let $S_p = S_p - \{e\}$.
        end
end.
```

The condition *check1(e)* is *true* if: *There is a clause in P with a body atom whose predicate is 'e'.*

The condition *check2* is *true* if: *There is no clause in P with an EDB atom E_1 in its body which unifies with an EDB atom E_2 in the head of the same clause.*

Example 3. Consider the following datalog program $\mathcal{P}$

$$\begin{array}{ll}
(1) & p(X) :- e(X,Y), r(Y) \\
(2) & p(X) :- e(Y,X), s(X) \\
(3) & p(X) :- e(X,X)
\end{array}$$

and assume that there are two materialized views v_1 and v_2 available

$$\begin{array}{ll}
(4) & v_1(X,Y) :- r(X), s(Y) \\
(5) & v_2(X,Y) :- e(X,Y) \\
(6) & v_2(X,Y) :- e(Y,X)
\end{array}$$

Applying the procedure described in the previous section, we get $\mathcal{P}_{disj}$ which contains the rules (1)-(3) and the following $\mathcal{V}^{-1}$-rules:

$$\begin{array}{ll}
(7) & r(X) :- v_1(X,Y) \\
(8) & s(Y) :- v_1(X,Y) \\
(9) & e(X,Y) \vee e(Y,X) :- v_2(X,Y)
\end{array}$$

Now we apply the procedure described above in order to obtain a datalog program which has the same atomic results as P_{disj}.

We unfold (1) at '$e(X,Y)$' using (9). We get a new equivalent program, which contains the clauses $P_1 = \{2,3,7,8,9,10,11,12\}$, where:

$$\begin{array}{ll}
(10) & p(X) \vee e(Y,X) :- v_2(X,Y), r(Y) \\
(11) & p(X) \vee e(Y,X) :- v_2(Y,X), r(Y) \\
(12) & p(X) \vee p(Y) :- v_2(Y,X), r(X), r(Y)
\end{array}$$

Unfolding (2) at '$e(X,Y)$' using (9), (10), (11) we get $P_2 = \{3,7,8,9,10,11,12,13,14,15,16,17\}$, where:

$$\begin{array}{ll}
(13) & p(X) \vee e(X,Y) :- v_2(Y,X), s(X) \\
(14) & p(X) \vee e(X,Y) :- v_2(X,Y), s(X) \\
(15) & p(X) \vee p(Y) :- v_2(Y,X), s(X), s(X) \\
(16) & p(X) :- v_2(X,Y), r(Y), s(X) \\
(17) & p(X) :- v_2(Y,X), r(Y), s(X)
\end{array}$$

Finally, unfolding (3) at '$e(X,X)$' using (9), (10), (11), (13), (14) we get $P_3 = \{7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24\}$, where:

$$\begin{array}{ll}
(18) & p(X) \vee e(X,X) :- v_2(X,X) \\
(19) & p(X) \vee e(X,X) :- v_2(X,X) \\
(20) & p(X) :- v_2(X,X) \\
(21) & p(X) :- v_2(X,X), r(X) \\
(22) & p(X) :- v_2(X,X), r(X) \\
(23) & p(X) :- v_2(X,X), s(X) \\
(24) & p(X) :- v_2(X,X), s(X)
\end{array}$$

Taking the factoring closure of the program, we get from clauses (12) and (15) the clauses:

$$
\begin{array}{ll}
(25) & p(X) :- \ v_2(X,X), r(X), r(X) \\
(26) & p(X) :- \ v_2(X,X), s(X), s(X)
\end{array}
$$

which are also added to the program. Clauses (19), (22), (24), (25) can be deleted since they are variants of other program clauses. Clauses (18), (21), (23) and (26) can be deleted since they are subsumed by clause (20). Clauses (9), (10), (11), (13) and (14) can be deleted because of lemma 4 (i.e. the atom with the predicate 'e' does not occur in any program clause). Finally, clauses (12) and (15) can be deleted from the program because of lemma 5. Therefore, the program obtained so far is $\{7, 8, 16, 17, 20\}$. Now, unfolding the EDB atoms in the bodies of (16) and (17) and deleting the redundant clauses we finally get:

$$
\begin{array}{l}
p(X) :- \ v_2(X,X) \\
p(Y) :- \ v_2(X,Y), v_1(Z_1,Y), v_1(X,Z_2) \\
p(X) :- \ v_2(X,Y), v_1(Z_1,X), v_1(Y,Z_2)
\end{array}
$$

This program is a retrievable Datalog program maximally contained in $\mathcal{P}$. $\square$

Concerning the procedure described above we can prove the following theorem.

Theorem 3. *The following hold for the procedure above:*

1. *The procedure always terminates.*
2. *Suppose that by applying the procedure above to a program P_{disj} we get a program P_{der} whose clause bodies do not contain any EDB atom. Then P_{der} is a non-recursive datalog($\neq$) program which is maximally contained in P.*

In the following we give a syntactic sufficient condition for the algorithm to end up with a datalog($\neq$) program.

Sufficient condition: *Consider the $\mathcal{V}^{-1}$-rules and the query program $\mathcal{P}$. For each rule in $\mathcal{P}$ there is at most one atom in its body whose predicate occurs in the head of a (non-trivial) disjunctive $\mathcal{V}^{-1}$-rule.*

Theorem 4. *If the sufficient condition holds for a a set of $\mathcal{V}^{-1}$-rules and a query program $\mathcal{P}$ then by applying the procedure to $\mathcal{V}^{-1} \cup \mathcal{P}$ we get a retrievable maximally contained non-recursive datalog($\neq$) program.*

5 Recursive Views

In the case where the views are given by recursive datalog programs and the query by a non-recursive datalog program $\mathcal{P}$, the problem of computing a retrievable program maximally contained in $\mathcal{P}$ is reduced to that of non-recursive view definitions (following lemma).

Lemma 6. *Let $\mathcal{P}$ be any non-recursive datalog program and $v_1, \dots, v_n$ be views given by any datalog programs over the EDB predicates of $\mathcal{P}$. Then, there exists a positive integer N depending only on the sizes of the programs so that the following holds:*

Consider a new set of views $v_1', \dots v_m'$ with view definitions $\mathcal{P}_{v_1'}, \mathcal{P}_{v_2'}, \dots, \mathcal{P}_{v_m'}$ respectively: where $\mathcal{P}_{v_i'}$ is the disjunction of all conjunctive queries that are produced by $\mathcal{P}_{v_i}$ and have size less that N.

If $\mathcal{P}_v$ is a retrievable program contained in $\mathcal{P}$ under the new view definitions, then $\mathcal{P}_v$ is a retrievable program contained in $\mathcal{P}$ under the original view definitions.

Proof. Let c_v be a conjunctive query generated by $\mathcal{P}_v$. Let c_v^{expi} be all expansions of c_v that result after replacing each occurrence of a view atom by one of their new definitions. Then, for each c_v^{expi}, there is a conjunctive query c_i generated by $\mathcal{P}$ and there exists a containment mapping, h_i, from c_i to c_v^{expi}.

Now, obtain all expansions oc_v^{expi}, $i = 1, 2, \ldots,$ of c_v considering the original view definitions. Suppose there exists an i such that there is no conjunctive query in $\mathcal{P}$ with a containment mapping on oc_v^{expi}. Then, a view occurrence in c_v has been replaced by a long ($> N = O(2^{s \log s})$, where s is the maximum size of the programs defining the query and the views) conjunctive query $R(v)$; $R(v)$ appears as a subquery in oc_v^{expi}. From oc_v^{expi}, we construct a shorter expansion oc_v^{expj}: We can pump $R(v)$ to get a shorter view expansion which, though, preserves all the neighborhoods of size $\leq m$ =the size of the maximum conjunctive query in $\mathcal{P}$ (see lemmata 1, 2). We apply this pumping on all expansions that are longer than N, getting, in the end, an expansion, c_v^{expi}, of c_v under the new view definitions. There exists, though, a conjunctive query of $\mathcal{P}$ with a containment mapping on c_v^{expi}. Since oc_v^{expi} and c_v^{expi} have the same neighborhoods of size up to m, there is a containment mapping also on oc_v^{expi}; this is a contradiction. □

Theorem 5. *Let $\mathcal{P}$ be any non-recursive connected datalog program and $v_1, \dots, v_n$ be views given by any datalog programs over the EDB predicates of $\mathcal{P}$. Suppose there exists a retrievable datalog program maximally contained in $\mathcal{P}$ which has no persistencies. Then, there exists a retrievable datalog program maximally contained in $\mathcal{P}$ which is non-recursive.*

Proof. An immediate consequence of the preceding lemma, and theorem 2. □

6 Chain programs

A *chain rule* is a rule over only binary predicates; moreover the first variable in the head is identical to the first variable of the first predicate in the body, the second variable in the head is identical to the last variable of the last predicate in the body and the second variable of the i-th predicate is identical to the first variable of the $i + 1$-th predicate in the body. A *chain program* contains only chain rules. It is easy to see that any conjunctive query generated by a chain program $\mathcal{P}$ corresponds to a canonical database which is a simple path spelling a word over the alphabet of the EDB predicate symbols. Thus, we can view a chain program as generating a language over this alphabet; we call this language $L_{\mathcal{P}}$. Observe that for chain programs, $L_{\mathcal{P}}$ is a context free language.

In the case, though, where both the query and the views are defined by recursive programs, it becomes undecidable to answer the question whether there is a non-empty datalog program that is contained in $\mathcal{P}$ and uses only the given views as EDB predicates; it remains undecidable even in the case where both program $\mathcal{P}$ and the views are given by chain programs.

The reduction, in the following theorem is done from the containment problem of context free grammars [17, 11]

Theorem 6. *Given a datalog chain program $\mathcal{P}$ and views $v_1, v_2, \ldots, v_m$ which are also given by chain programs over the EDB predicates of $\mathcal{P}$, it is undecidable whether there is a non-empty retrievable datalog program $\mathcal{P}_v$ contained in $\mathcal{P}$.*

In some cases, though, we can have a negative answer in the question whether there exists a "simple" non–empty retrievable program, as it is stated in the theorem that follows.

Here on, we consider the following situation: A datalog query given by chain program $\mathcal{P}$ and materialized views, $v_1, v_2, \ldots, v_m$, over the EDB predicates of $\mathcal{P}$, which are also given by chain programs. Let $\mathcal{P}_v$ be a datalog program (not necessarily chain) contained in $\mathcal{P}$ that uses only $v_1, v_2, \ldots, v_m$ as EDB predicates.

Consider a datalog program $\mathcal{P}$ where both IDB and EDB predicates are binary. Take any conjunctive query c produced by $\mathcal{P}$ and take the canonical database of c. We call $\mathcal{P}$ *simple* if, for all c, the canonical database (viewed as a directed graph) contains a number of pairwise node-disjoint simple paths with endpoints on the two head constants, i.e., besides the head constants every constant (viewed as a node) has in–degree one and out–degree one. Chain programs are simple.

We need some technical concepts here on pumping lemmas for formal languages. Let w be a word over the alphabet Σ. For fixed positive integer N, we say that $w_i = ux^ivy^iw, i = 1, 2, \ldots$ is a *pumping sequence* for w if $w = uxvyw$ and $\mid xy \mid < N$. For context free languages, there exists a positive integer N, such that, for any word in a language, L, longer than N, there exists a pumping sequence $w_i, i = 1, \ldots$, such that $w_i, i = 1, \ldots$ also belongs in the language. We call such a pumping sequence a *proper* pumping sequence *wrto* language L.

Theorem 7. *Let $\mathcal{P}$ be a chain program and $v_1, v_2, \ldots, v_m$ views over the EDB predicates of $\mathcal{P}$, which are also given by chain programs $V_1, V_2, \ldots, V_m$, respectively. Suppose there exists a simple retrievable datalog program contained in $\mathcal{P}$. Then there exists a fixed positive integer N (depending only on the sizes of the programs for the views and $\mathcal{P}$), and there is a view definition V_i such that for any word w in L_{V_i}, with $\mid w \mid > N$, the following happens: There is a word $w_{\mathcal{P}}$ in $L_{\mathcal{P}}$ such that $w_{\mathcal{P}} = ww'$ and for any proper pumping sequence, $w_1, w_2, \ldots$, of w wrto L_{V_i}, there exists an infinite subsequence $w_{i_1}, w_{i_1+1}, \ldots$ such that $w_{i_1+j}w'$ also belongs to $L_{\mathcal{P}}$, for all $j = 1, 2, \ldots$.*

For an application, consider the following program $\mathcal{P}$:

$$
\begin{aligned}
&p(X,Y) :- a(X,Z_1), p(Z_1,Z_2), p(Z_2,Z_3), a(Z_3,Y)\\
&p(X,Y) :- b(X,Y)
\end{aligned}
$$

and the views:

$$
\begin{aligned}
&v_1(X,Y) :- a(X,Z_1), v_1(Z_1,Z_2), a(Z_2,Y)\\
&v_1(X,Y) :- b(X,Z_1), b(Z_1,Y)\\
&v_2(X,Y) :- a(X,Z_1), v_2(Z_1,Y)\\
&v_2(X,Y) :- a(X,Y)
\end{aligned}
$$

Theorem 8. *Given the views v_1, v_2, there is no retrievable simple datalog program contained in $\mathcal{P}$.*

Proof. Consider any word $w = w'w''$ of $L_{\mathcal{P}}$ where w' is a sufficiently large word of L_{v_1} (L_{v_2} respectively). Consider a proper pumping sequence of w' wrto L_{v_1} (L_{v_2} respectively), $w_1, w_2, \ldots$. Suppose there exists a retrievable simple program. Then, for infinitely many i's, $w_i w'$ is also a word in $L_{\mathcal{P}}$, according to the theorem above. Observe, though, that any word in $L_{\mathcal{P}}$ longer than four contains $2k+2$ as and $k+2$ bs. $w_i w'$ will not retain this balance, therefore it is not a word of $L_{\mathcal{P}}$. □

7 Conclusion

We investigated the problem of answering queries using materialized views. In particular, we searched for a retrievable program maximally contained in the query. In the case the query is defined by a non-recursive datalog program and the views by recursive datalog programs, we reduced the problem to that of non-recursive definitions for both the query and the views. We showed that, in the case where both the query and the views are defined by recursive datalog programs, then the problem becomes easily undecidable; we showed, though, some methods to produce negative results. It would be interesting to further pursue this latter line of research. In the case both the query and the views are defined by non-recursive datalog programs, we showed how, in certain cases, we can produce a retrievable non-recursive datalog($\neq$) program maximally contained in the query. It seems that, further investigation towards this direction will produce interesting results. We are currently working on this.

Acknowledgment: We thank Vassilis Vassalos for helpful discussions.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. ACM Symposium on Principles of Database Systems*, 1998.

[2] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the Sixth International Conference on Database Theory*, pages 1–18. Springer-Verlag, 1997.

[3] F. Afrati, S. Cosmadakis, and M. Yannakakis. On datalog vs. polynomial time. *J. Computer and Systems Sciences*, 51(2):117–196, 1995.

[4] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseak Shim. Optimizing queries with materialized views. In *Proceedings of the 11th International Conference on Data Engineering, Los Alamitos, CA*, pages 190–200. IEEE Comput. Soc. Press, 1995.

[5] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. In *Proceedings of the Sixth International Conference on Database Theory*, pages 56–70. Springer-Verlag, 1997.

[6] Oliver M. Duschka. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, December 1997.

[7] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. 16th ACM SIGACT-SIGMOD-GIGART Symposium on Principles of Database Systems*, pages 109–116, 1997.

[8] Oliver M. Duschka and Michael R. Genesereth. Query planning with disjunctive sources. In *Proc. of the AAAI-98 Workshop on AI and Information Integration*, 1998.

[9] M. Gergatsoulis. Correctness-preserving transformations for disjunctive logic programs. Demo 97/2, Institute of Informatics & Telecom. NCSR 'Demokritos', 1997.

[10] Manolis Gergatsoulis. Unfold/fold transformations for disjunctive logic programs. *Information Processing Letters*, 62(1):23–29, April 1997.

[11] M. A. Harrison. *Introduction to formal language theory.* Addison-Wesley, 1978.

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

[13] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, 1995.

[14] Alon Y. Levy, Divest Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 1995. Special Issue on Networked Information Discovery and Retrieval.

[15] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.

[16] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. 14th ACM SIGACT-SIGMOD-GIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[17] Oded Shmueli. Decidability and expressiveness aspects of logic queries. In *Proc. 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 237–249, 1987.

[18] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I & II. Computer Science Press, 1989.

[19] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory*, pages 19–40. Springer-Verlag, 1997.

[20] Jeffrey D. Ullman and Allen Van Gelder. Parallel complexity of logical query programs. In *Proc. 27th IEEE Symp. on Foundations of Comp. Sci.*, pages 438–454, 1986.