Certain Answers for Views and Queries Expressed as Non-recursive Datalog Programs with Negation

Victor Felea

"Al.I.Cuza" University of Iasi
Computer Science Department, 16 General Berthelot Street, Iasi, Romania
felea@infoiasi.ro
http://www.infoiasi.ro

Abstract. In this paper, we study the problem to compute certain answers in case when view definitions are expressed as non-recursive datalog programs with negation and queries are expressed as semi-positive non-recursive datalog programs with negation. Two situations are analyzed: the open world assumption (OWA) and the close world assumption (CWA). Associated to a view, and an extension of the view, a tree is constructed, which is useful to specify a method to compute certain answers.

Keywords: Datalog program, negation, query, view, certain answer.

1 Introduction

View-based query processing is a problem of computing the answer to a query based on a set of views. This problem is very important in application areas, such as query optimization, data warehousing, data integration. Two basic approaches to view-based query processing are known: query-rewriting and query-answering. These approaches are discussed in [4]. In the second approach, so-called certain tuples are computed. The problem of query-answering is the following: given a query on a database schema, and a set of views over the same schema, can we answer the query using only the answers to the views? This problem was intensively studied in the literature. Thus, in [1] the authors give the complexity of the problem of answering views using materialized views, where the languages for view definitions and queries are: conjunctive queries with inequalities, positive queries, datalog program and first order logic. In [12], some applications of the problem of answering queries using views, and algorithms are specified. The problem of view-based query processing in the context where databases are semistructured and both the query and the view are expressed as regular path queries, are studied in [6] and [7]. In [4], the authors analyze the complexity of query answering in the presence of key and inclusion dependencies. The answering query problem, in case when queries and views are in conjunctive form with arithmetic comparisons, is studied in [2]. In [13], the authors define so-called

N.T. Nguyen, C.-G. Kim, and A. Janiak (Eds.): ACIIDS 2011, LNAI 6591, pp. 67–77, 2011.

"relative containment", which formalizes the notion of query containment relative to the source that occur to in a data-integration systems. The problem of answering queries, using materialized views in the presence of negative atoms in view, is studied in [7]. In [8], the authors give a system that computes consistent answers to Datalog(disjunctive logic programming) queries with stable models semantics. A semantic model to compute consistent query answers is given in [9]. In [3], the authors apply logic programming based on answer sets to the retrieving consistent information from a possible inconsistent database. The problem of answering datalog queries using views is undecidable ([10]). In [14], the problem of whether a query Q can be answered using a set of views is studied.

Concerning the certain answers, the complexity of finding certain answers is discussed in [1], considering the case when views and queries are expressed in languages: conjunctive form, conjunctive form with inequalities, non-recursive datalog, datalog, first-order formula. In case when the query is expressed in datalog, and does not contain comparison predicate, and the views are in conjunctive form, the set of certain answers can be obtained using so-called query plan, which is a datalog program where extensional relations are the source relations [10]. The query plan that produces all certain answers is called the maximally-contained plan, and is defined in [10]. In [11], the authors show that the problem of computing the certain answers for union of conjunctive queries with inequalities is in $coNP$. In the paper [6], the authors study the problem of answering a query based on precomputed answers (that can be certain answers) for a set of views, in the context of Description Logic. By our best knowledge, the problem to compute the certain answer set in case when the negation occurs in views or query was not considered in literature up to now.

In this paper, we study the problem of computing certain answers in case when the views are expressed by non-recursive datalog programs with negation and the query is a non-recursive semi-positive datalog program.

2 Basic Definitions and Notations

Let Dom be a countable infinite domain for databases. The elements of Dom are called constants. Let $\mathcal{V}$ be a view expressed as a non-recursive datalog program having $V(\overline{x})$ as the head of a rule that is called the main rule. Each rule from $\mathcal{V}$ can contain negated literals. Let us denote by $EDB(\mathcal{V})$, $IDB(\mathcal{V})$, and $Rel(\mathcal{V})$ the set of extensional, intensional symbols, all symbols from $\mathcal{V}$, respectively. A rule that has $f(\overline{z})$ as its head is called a definition of f. Assume that for each $f \in IDB(\mathcal{V})$, there exists a single definition of f. The rules can contain free variables (these variables are those from the head of rule), existentially quantified variables (these variables occur in the rule body), constants that appear in the body of the rule. There are two restrictions about variables or constants that occur in the definition of an intensional symbol. The first one is: each variable that occurs in the head of rule it also must appear in the positive part of the body of rule. This is called the safe property of the view definition. The second one: each variable or constant that occurs in the negated part of the definition

of f, must occur in its positive part. This property is called "safe negation". A query Q is considered as a non-recursive semi-positive datalog program (the semi-positive datalog program is a datalog program where each definition symbol contains in its negated part only extensional symbols). The rules of Q satisfy the restrictions like as those for $\mathcal{V}$. Assume that $EDB(Q) \subseteq EDB(\mathcal{V})$. In the following definition we give the notion of certain answer.

Definition 1. *Let Q be a query expressed as a non-recursive semi-positive datalog program and $\mathcal{V}$ a view expressed as a non-recursive datalog program, having $V(\overline{x})$ as the head of the main rule. Let $I = \{\overline{w}_1, \ldots, \overline{w}_m\}$ be an extension of the view $\mathcal{V}$. The tuple $\overline{t}$ having $arity(\overline{t}) = arity(\overline{x})$, is a certain answer for I, $\mathcal{V}$, and Q under OWA, if $t \in Q(D)$ for all databases D defined on Dom such that $I \subseteq \mathcal{V}(D)$, and $Rel(D) \subseteq Rel(\mathcal{V})$. The tuple $\overline{t}$ is a certain answer for I, $\mathcal{V}$, and Q under CWA, if $\overline{t} \in Q(D)$ for all databases D defined on Dom such that $I = \mathcal{V}(D)$, and $Rel(D) \subseteq Rel(\mathcal{V})$.*

Intuitively, a tuple is a certain answer of the query Q, if it is an answer for any of the possible database instances which are consistent with the given extensions of the view. In the case OWA, the relation $I \subseteq \mathcal{V}(D)$ is equivalent to $\overline{w}_i \in V(D)$, for each $i, 1 \leq i \leq m$. Assume that all values from I belong to Dom. Let us denote by C the values from I, and the constants from $\mathcal{V}$. Let Y be the set of all variables that are existentially quantified in the rules of $\mathcal{V}$. Let π be a partition on the set $C \cup Y$, and $Class_\pi$ the set of all classes defined by π. A partition π is called a $C - partition$, if for two distinct constants c_1 and c_2, we have $[c_1]_\pi \neq [c_2]_\pi$, hence two distinct constants occur in two distinct classes with respect to π. Through the paper we use only $C - partitions$, so in the paper when we write *partition*, we mean $C - partition$. For a partition π defined on $C \cup Y$, we consider a mapping from $C \cup Y$ into $Class_\pi$, denoted η_π and defined as follows: $\eta_\pi(t) = [t]_\pi$, where $[t]_\pi$ denote the class that contains t. The mapping η_π is naturally extended for a vector $\overline{w} = (t_1, \ldots, t_r)$ having the components from $C \cup Y$, by $\eta_\pi(\overline{w}) = (\eta_\pi(t_1), \ldots, \eta_\pi(t_r))$. For an atom $R(\overline{w})$, we consider $\eta_\pi(R(\overline{w})) = R(\eta_\pi(\overline{w}))$. For a set S of atoms having the form $R(\overline{w})$, we define $\eta_\pi(S) = \{\eta_\pi(R(\overline{w}))|R(\overline{w}) \in S\}$. For a database D defined on Dom, we consider $val(D)$ the set of all values that occur in the atoms of D. Formally, $val(D) = \{v|\exists R(\overline{w}) \in D, v \text{ is a component of } \overline{w}\}$. The view $\mathcal{V}$ is said consistent with I under OWA if there exists a database D over Dom such that $I \subseteq \mathcal{V}(D)$. Let us denote by $f_1 o f_2$ the composition of the two mappings f_1, f_2, where $(f_1 o f_2)(t) = f_2(f_1(t))$. Now, we need to define a formula corresponding to a view definition. Let f be from $IDB(\mathcal{V})$ and its definition having the form:

$$f(\overline{z}) : -S_1(\overline{z}, \overline{t}_1), \ldots, S_n(\overline{z}, \overline{t}_n), \neg S_{n+1}(\overline{z}, \overline{t}_{n+1}), \ldots, \neg S_{n+p}(\overline{z}, \overline{t}_{n+p}) \qquad (1)$$

The vector $\overline{z}$ contains all free variables from this definition, the vector $\overline{t}_j$ contains all existentially quantified variables from $S_j(\overline{z}, \overline{t}_j)$. Let us denote by $\overline{y}_j$ the vector of all variables that occur in $S_j(\overline{z}, \overline{t}_j)$. Let $S_{n+j}(\overline{y}_{n+j})$ be an atom that occurs in the negated part of $f(\overline{z})$. If the symbol S_{n+j} occurs in the positive part of f with indexes $\alpha_1, \ldots, \alpha_q$, then we have: S_{n+j}=S_{α_i}, for each $i, 1 \leq i \leq q$,

and $S_{n+j} \neq S_\beta$, for each $\beta \in \{1,\ldots,n\} - \{\alpha_1,\ldots,\alpha_q\}$. We associate to the atom $S_{n+j}(\overline{y}_{n+j})$ a formula denoted ϕ^j, and defined as follows: $\phi^j = (\overline{y}_{n+j} \neq \overline{y}_{\alpha_1}) \wedge \ldots \wedge (\overline{y}_{n+j} \neq \overline{y}_{\alpha_q})$, where the expression $(\overline{y}_l \neq \overline{y}_s)$ denotes the following disjunction: $(y_l^1 \neq y_s^1) \vee \ldots \vee (y_l^r \neq y_s^r)$, the tuples $\overline{y}_l$ and $\overline{y}_s$ having the form: $\overline{y}_l = (y_l^1,\ldots,y_l^r)$, $\overline{y}_s = (y_s^1,\ldots,y_s^r)$. In case when S_{n+j} does not occur in the positive part of f, then we consider $\phi^j = TRUE$.

Definition 2. *The formula ϕ corresponding to f, denoted $\phi(f)$ is the conjunction of all formulas ϕ^j, $1 \leq j \leq p$, that means: $\phi(f) = \phi^1 \wedge \ldots \wedge \phi^p$.*

Now, we formally define the logic value of a formula for a given partition.

Definition 3. *Let π be a partition defined on $C \cup Y$ and $\phi(f)$ the formula constructed for f as we mentioned above. The logic value of $\phi(f)$ for π, denoted $\pi(\phi(f))$ is recursively defined as follows:*
(i) If $\phi \equiv (t \neq t')$, where t and $t' \in C \cup Y$, then $\pi(\phi) = TRUE$ if there is no class E from $Class_\pi$ such that $t, t' \in E$, i.e. $[t]_\pi \neq [t']_\pi$.
(ii) $\pi(\phi_1 \wedge \phi_2) = \pi(\phi_1) \wedge \pi(\phi_2)$, $\pi(\phi_1 \vee \phi_2) = \pi(\phi_1) \vee \pi(\phi_2)$.

3 The Construction of $TREE(I, \mathcal{V})$

Firstly, we need to give a definition that precises some notions which will be used in the construction of a tree corresponding to I and $\mathcal{V}$.

Definition 4. *Let π be a partition, T a database defined on $Class_\pi$, and f an intensional symbol from $\mathcal{V}$. Let I' be the projection of T on f, i. e. $I' = T[f]$. Let $I' = \{f(\overline{z}_1),\ldots,f(\overline{z}_h)\}$ and $f(\overline{z}) :- f_1(\overline{z},\overline{t})$ the definition of f in $\mathcal{V}$, where $\overline{t}$ contains all existentially quantified variables from the right-hand part of the definition of f. We denote by $pos(f(\overline{z}))$, $(neg(f(\overline{z})))$, the set of all positive (negated) atoms from the definition of $f(\overline{z})$. We consider the substitutions of the vector $\overline{z}$ with $\overline{z}_j$, $1 \leq j \leq h$ in the view definition of f and for two distinct indexes j and l we take the existentially quantified variable sets disjoint.*
(a) We denote these rules by $Ext(f, I')$, i.e.

$$f(\overline{z}_j) :- f_1(\overline{z}_j, \overline{t}_j), 1 \leq j \leq h.$$

(b) The set of all existentially quantified variables from $Ext(f, I')$ will be denoted by $ExtVar(f, I')$, i.e. $ExtVar(f, I') = \cup_{1 \leq j \leq h} \overline{t}_j$.
(c) Two databases defined on $Class_\pi$, denoted $DPos(f, I')$ and $DNeg(f, I')$, and specified as follows:
$DPos(f, I') = \cup_{1 \leq j \leq h} pos(f(\overline{z}_j))$, $DNeg(f, I') = \cup_{1 \leq j \leq h} neg(f(\overline{z}_j))$.

Associated to I and $\mathcal{V}$, we construct a tree denoted $TREE(I, \mathcal{V})$, where its root is denoted by α_0. For each node α of the tree, we associate so-called "basic elements" and so-called "calculated elements". The "basic elements" are:

(a) A set of intensional symbols from $\mathcal{V}$, denoted $RInt(\alpha)$,
(b) A set of constants and variables, denoted $OldCV(\alpha)$,

(*c*) A partition defined on $OldCV(\alpha)$, denoted $\pi(\alpha)$, and
(*d*) A database defined on $Class_{\pi(\alpha)}$, denoted $OldD(\alpha)$. These elements are also called inherited, because they are calculated for the precedent node.

The "calculated elements" associated to α, using Definition 4 are the following:
(*e*) For each symbol f from $RInt(\alpha)$ we compute $I' = OldD(\alpha)[f]$, $E1 = Ext(f, I')$, $D1 = DPos(f, I')$, $D2 = DNeg(f, I')$, $E2 = ExtVar(f, I')$.
(*f*) A new set of constants and variables $NewCV(\alpha) = OldCV(\alpha) \cup E2$,
(*g*) A new database defined on $Class_{\pi(\alpha)}$, $NewD(\alpha) = (OldD(\alpha) - I') \cup D1$.
(*h*) A database defined on $Class_{\pi(\alpha)}$ containing all atoms from all nodes γ situated on the path from α_0 to α, $DTNeg(\alpha) = DTNeg(\alpha') \cup D2$, where α' is the immediate predecessor of α.
(*i*) The set of all partitions defined on $NewCV(\alpha)$, that are extensions of $\pi(\alpha)$. This set is denoted $Partition(\alpha)$.
(*j*) For each partition π_1 from $Partition(\alpha)$, we compute the set $\mathcal{M}_{\pi_1}$, specified in Definition 5.
(*k*) A variable $TERM(\alpha)$ having the values: 0 when α is not terminal, 1 in case when α is terminal, but $RInt(\alpha) \neq \emptyset$, 2 when the node is terminal, but the database $OldD(\alpha)$ is inconsistent with $\mathcal{V}$, 3 otherwise.

Definition 5. *Let π be a partition, T a database defined on $Class_\pi$, $f \in IDB(\mathcal{V})$, $I' = T[f]$, $NewCV$ a set of constants and variables, π_1 a partition defined on $NewCV$, that is an extension of π, and $Ext(f, I')$ the datalog program specified in Definition 4. We define two databases on $Class_{\pi_1}$ denoted $T^{min}_{\pi_1}$ and $T^{max}_{\pi_1}$ and defined as follows:*

$$T^{min}_{\pi_1} = \eta_{\pi_1} DPos(f, I'), \tag{2}$$

$$T^{max}_{\pi_1} = \{\eta_{\pi_1} R(\overline{w}), R \in Rel(DPos(f, I')) \text{ and } \overline{w} \text{ has components from } NewCV\} - \eta_{\pi_1} DTNeg(\alpha) \tag{3}$$

A set of databases defined on $Class_{\pi_1}$, denoted $\mathcal{M}_{\pi_1}$ will be defined as follows:

$$\mathcal{M}_{\pi_1} = \{T_1 | T^{min}_{\pi_1} \subseteq T_1 \subseteq T^{max}_{\pi_1}\} \tag{4}$$

Corresponding to the root α_0 we take the following: $RInt(\alpha_0) = \{V\}$, $OldCV(\alpha_0) = C$, $\pi(\alpha_0)$ is the discrete partition on C, $OldD(\alpha_0) = \{V(\overline{w}_1), \ldots, V(\overline{w}_m)\}$, where $I = \{\overline{w}_1, \ldots, \overline{w}_m\}$, and $DTNeg(\alpha_0) = \emptyset$. Now, we specify some details about the construction of $TREE(I, \mathcal{V})$. Suppose we have constructed the node α having the elements described above. For each tuple having the form (f, π_1, T_1), where $f \in RInt(\alpha)$, $\pi_1 \in Partition(\alpha)$, and $T_1 \in \mathcal{M}_{\pi_1}$, we construct an immediate successor of α, denoted β, such that: $RInt(\beta) = RInt(\alpha) - \{f\} \cup IDB(Ext(f, I'))$, $OldCV(\beta) = NewCV(\alpha)$, $\pi(\beta) = \pi_1$, $OldD(\beta) = T_1$, $DTNeg(\beta) = DTNeg(\alpha) \cup DNeg(f, I')$.

In case when the view definition of f does not contain existentially quantified variables ($ExtVar(f, I') = \emptyset$), then the immediate successors of α are constructed for each vector (f, π_1, T_1) , where $f \in RInt(\alpha)$, $\pi_1 = \pi(\alpha)$ and $T_1 \in \mathcal{M}_{\pi_1}$. In case when the node α is terminal, but $RInt(\alpha) \neq \emptyset$, then we

called this node "failed node". In case when α is terminal and $RInt(\alpha) = \emptyset$, then it is need to test this node for consistency. Let $D = OldD(\alpha)$. For each $f \in IDB(\mathcal{V})$, we compute the answer of f for D, denoted $f(D)$. The test for this node α is the following:

$$(\cup_{f \in IDB(\mathcal{V})} f(D)) \cap DTNeg(\alpha) \neq \emptyset \vee DTNeg(\alpha)[EDB(\mathcal{V})] \cap OldD(\alpha) \neq \emptyset \quad (5)$$

In case when the node α satisfies this condition, then $TERM(\alpha) = 2$ otherwise $TERM(\alpha) = 3$. We remark that the answer of f for D is computable because the datalog program $\mathcal{V}$ is non-recursive. One can specify a recursive procedure denoted $CONSTR$, which for a given node α constructs all successors of α in $TREE(I, \mathcal{V})$. The parameters of the procedure $CONSTR$ are α, and those specified in $(a) - (d)$, and $DTNeg$. The main program to construct $TREE(I, \mathcal{V})$ could be the following:

BEGIN
$C1 = C$; $T = \{V(\overline{w}_1), \ldots, V(\overline{w}_m)\}$; π_0 is the discrete partition on $C1$;
$RInt = \{V\}$; $DTNeg = \emptyset$; $TERM(\alpha_0) = 0$;
CALL $CONSTR(\alpha_0, RInt, C1, \pi_0, T, DTNeg)$;
END

Example 1. Let us define a view $\mathcal{V}$ by the following two rules, and $I = (0)$ an extension of $\mathcal{V}$:
$V(x) : -f(x, z_1), \neg f(z_1, x)$, $f(t_1, t_2) : -R(t_1, z_2), R(z_2, t_2), \neg R(t_1, t_1)$.
The $TREE(I, \mathcal{V})$ will have three levels (the root is considered on level 1). For its root α_0 we have: $RInt(\alpha_0) = \{V\}$, $OldCV(\alpha_0) = \{0\}$, $\pi(\alpha_0) = \{\overline{0}\}$, $OldD(\alpha_0) = V(0)$. In this example we denote by $\overline{t_1 \ldots t_h}$ the class that contains the elements t_1, ...,t_h. We have: $f = V$, $I' = V(0)$, $Ext(f, I')$ consists of the rule: $V(0) : -f(0, z_1), \neg f(z_1, 0)$, $DPos(f, I') = f(0, z_1)$, $DNeg(f, I') = f(z_1, 0)$,$ExtVar(f, I') = \{z_1\}$, $NewD(\alpha_0) = \{f(0, z_1)\}$, $NewCV(\alpha_0) = \{0, z_1\}$, $DTNeg(\alpha_0) = \{f(z_1, 0)\}$. There are two partitions defined on $NewCV(\alpha_0)$, namely: $\pi_1 = \{\overline{0}, \overline{z_1}\}$ and $\pi_2 = \{\overline{0z_1}\}$. The formula ϕ associated to the extension $Ext(f, I')$ is $\phi = (z_1 \neq 0)$. This formula is satisfied only by π_1. For π_1, we compute the set $\mathcal{M}_{\pi_1}$ and if we denote by $M_1 = \{f(\overline{0}, \overline{0}), f(\overline{z_1}, \overline{z_1})\}$, and by $\mathcal{P}(M_1)$ the set of all subsets from M_1, then $\mathcal{M}_{\pi_1} = \{\{f(0, z_1)\} \cup S | S \in \mathcal{P}(M_1)\}$. Thus the node α_0 has four immediate successors. For $S = \emptyset$, let us denote this node by β_1. This node will be extended with two successors, denoted β_{11} and β_{12}, for $RInt = f$, and the partition $\pi_3 = \{\overline{0}, \overline{z_1 z_2}\}$. For $RInt = \{f\}$, and the partition $\pi_4 = \{\overline{0}, \overline{z_1}, \overline{z_2}\}$, we obtain 64 successors. For the nodes β_{11} and β_{12}, we have $TERM(\beta_{11}) = 3$, but $TERM(\beta_{12}) = 2$. In a similar manner, we obtain the elements associated to other nodes from $TREE(I, \mathcal{V})$.

4 Some Properties of Nodes from $TREE(I, \mathcal{V})$

In the following, we point out some properties of databases associated to the nodes from $TREE(I, \mathcal{V})$.

Proposition 1. *Let α be a node from $TREE(I, \mathcal{V})$ having $TERM(\alpha) \neq 3$. Let τ be an injective mapping from $Class_{\pi(\alpha)}$ into Dom, and $D' = \tau(OldD(\alpha))$. We have: $\mathcal{V}(D') = \emptyset$.*

In the following, we consider as mappings τ only those that preserve constants, i. e. $\tau([c]_\pi) = c$. We remark that the view $\mathcal{V}$ is consistent with I iff there exists a terminal node in $TREE(I, \mathcal{V})$ having $TERM(\alpha) = 3$. Now, we give a result about the relation between the answers of a query Q, expressed by a semi-positive non-recursive datalog program, for two databases defined on *Dom*.

Proposition 2. *Let D_1 and D be two databases defined on the schema $S = EDB(\mathcal{V})$ such that $D_1 \subseteq D$, and for each atom $R(\overline{w}) \in D - D_1$, there is a component v from $\overline{w}$ such that $v \notin val(D_1)$. Then, for each query Q expressed as a semi-positive non-recursive datalog program having $EDB(Q) \subseteq S$, we have $Q(D_1) \subseteq Q(D)$.*

Proof. Since the query Q is non-recursive, we can define for each relational symbol S from Q a level, denoted $level(Q)$, and define as follows:

(i) For each $S \in EDB(Q)$, we take $level(S) = 0$.
(ii) If f_j, $1 \leq j \leq s + t$ occur in the right part of the definition of f from (1), and $max\{level(f_j) | 1 \leq j \leq s + t\} = h$, then $level(f) = h + 1$.

We show by induction on $level(f)$ the statement: $f(D_1) \subseteq f(D)$. Firstly, let f be a symbol such that $level(f) = 1$, then from (ii) we have $level(f_j) = 0$, hence f_j are EDB-symbols. Let $\overline{w}$ be from $f(D_1)$. This implies there exists a substitution θ from the variables that occur in the definition of f into *Dom* such that $\theta f_j(\overline{z}_j) \in D_1$, $1 \leq j \leq n$ and $\theta f_{n+i}(\overline{z}_{n+i}) \notin D_1$, $1 \leq i \leq p$ and $\theta\overline{z} = \overline{w}$. We must show that: $\theta f_{n+i}(\overline{z}_{n+i}) \notin D$. Assume the contrary, then there exists i such that $\theta f_{n+i}(\overline{z}_{n+i}) \in D$. Since by the induction hypothesis $\theta f_{n+i}(\overline{z}_{n+i}) \notin D_1$, we obtain there exists a component v from $\theta\overline{z}_{n+i}$ such that $v \notin val(D_1)$. On the other hand, using the safeness property regarding the negation, we get: all components of $\theta\overline{z}_{n+i}$ belong to $\theta(\cup_{1 \leq j \leq n}\overline{z}_j)$, therefore all components of $\theta\overline{z}_{n+i}$ belong to $val(D_1)$, which is a contradiction. Thus, $w \in f(D)$.

For the inductive step, assume that $f_i(D_1) \subseteq f_i(D)$, for each symbol f_i, having $level(f_i) \leq h$. Let f be a symbol having $level(f) = h+1$, and f has the definition specified in (1). By the hypothesis of induction, we have $f_i(D_1) \subseteq f_i(D)$. Since the query Q is semi-positive, we have: $f_{n+i} \in EDB(Q)$, $1 \leq i \leq p$. As in case when $level(f) = 1$, we obtain $f(D_1) \subseteq f(D)$. □

Lemma 1. *Let $OldCV$ be a set of constants and variables, and D a database defined on Dom having $Rel(D) \subseteq Rel(\mathcal{V})$. Let θ be a substitution from $OldCV$ into Dom. Then, there exist a partition π defined on $OldCV$, an injective mapping τ_θ from $Class_\pi$ into Dom such that $\theta = \eta_\pi o \tau_\theta$.*

Proof. We define the partition π as follows: $t_1 \pi t_2$ iff $\theta(t_1) = \theta(t_2)$. The mapping τ_θ is as follows: $\tau_\theta([t]_\pi) = \theta(t)$, for each $t \in OldCV$. The statement $\theta = \eta_\pi o \tau_\theta$ results immediately. □

Proposition 3. *Let OldCV be a set of constants and variables, θ a substitution from OldCV into Dom. Let π be the partition of OldCV corresponding to θ (by Lemma 1). Let OldD be a database defined on* Class_π *and $f \in RInt(OldD)$. Let $I' = OldD[f] = \{f(\overline{z}_1), \ldots, f(\overline{z}_s)\}$. Let D be a database defined on Dom, having $Rel(D) \subseteq Rel(\mathcal{V})$. Let E be the extension of f and I', i.e. $E = Ext(f, I')$. Let $\phi(E)$ be the associated formula to E, and $V_1 = ExtVar(f, I')$, the set of all existentially quantified variables from E. We consider the definition of f expressed as in (1). Assume that there exists a substitution θ from OldCV into Dom such that $\theta f(\overline{z}_j) \in f(D)$, for each j, $1 \leq j \leq s$. Moreover, assume that $I \subseteq V(\tau_\theta(OldD))$, where τ_θ is the injective mapping corresponding to θ (Lemma 1). The the following statements hold:*

(i) There exists a substitution θ_1 from $NewCV = OldCV \cup V_1$ into Dom*, such that $\theta_1 S_k(\overline{z}_j, \overline{t}_{jk}) \in S_k(D)$, $1 \leq k \leq n$, $1 \leq j \leq s$. For two different indexes j and m the sets of existentially quantified variables are disjoint, i.e. $(\cup_{1 \leq l \leq n} \overline{t}_{jl}) \cap (\cup_{1 \leq l \leq n} \overline{t}_{ml}) = \emptyset$. Let $DPos(f, I') = \cup_{1 \leq k \leq n} \cup_{1 \leq j \leq s} S_k(\overline{z}_j, \overline{t}_{jk})$. Moreover, the substitution θ_1 is an extension of θ.*
(ii) $\theta_1 S_{n+e}(\overline{z}_j, \overline{t}_{jn+e}) \notin S_{n+e}(D)$, $1 \leq e \leq p$, $1 \leq j \leq s$.
(iii) The partition π_1 corresponding to the substitution θ_1 (Lemma 1) defined on $NewCV$ is an extension of π, and satisfies $\pi_1(\phi(E)) = TRUE$.
(iv) There exist a database $D' \subseteq D$, and a database T on Class_{π_1} *from $\mathcal{M}_{\pi_1}$ such that $D' = \tau_{\theta_1}(T)$.*
(v) The database D' specified above satisfies: $I \subseteq \mathcal{V}(D')$, and for each atom $S(\overline{w})$ from $D - D'$ the vector $\overline{w}$ contains at least a component that does not belong to V', where $V' = \theta_1(NewCV)$.

Proof. The statements (i) and (ii) follow from the relations $\theta f(\overline{z}_j) \in f(D)$, $1 \leq j \leq s$ and the definition of f. Since the substitution θ_1 is an extension of θ, it results that π_1 is an extension of π. Since the substitution θ satisfies the relations $\theta f(\overline{z}_j) \in f(D)$, we obtain that $\pi_1(\phi(E)) = TRUE$ (the statement (iii)). For the statements (iv) and (v) we consider a database defined on *Dom* as follows: $D' = \{S(\overline{v}) | S(\overline{v}) \in D,\ S \in Rel(NewD)$ and $\overline{v}$ contains only values from $V'\}$. The mapping τ_{θ_1} is bijective from $Class_{\pi_1}$ into V'. Let $T = \tau_{\theta_1}^{-1}(D')$. Let us show that $T \in \mathcal{M}_{\pi_1}$. From the statement (i), we obtain: $\theta_1 DPos(f, I') \subseteq D'$, which implies $T_{\pi_1}^{min} \subseteq T$, applying the substitution $\tau_{\theta_1}^{-1}$, and using the relation $\tau_{\theta_1}^{-1} o \theta_1 = \eta_{\pi_1}$. By Definition 5, we have: $T_{\pi_1}^{min} = \eta_{\pi_1} DPos(f, I')$, and $T_{\pi_1}^{max} = \{S(\overline{w}) | S \in Rel(DPos(f, I'))$, $\overline{w}$ has components from $Class_{\pi_1}\} - \eta_{\pi_1} DT$, where DT is $DNeg(f, I')$. We obtain that $T \subseteq T_{\pi_1}^{max}$, hence $T \in \mathcal{M}_{\pi_1}$.

To show the statement (v): By the relations specified in (i) and (ii), $V' = \theta_1(NewCV)$ and by the definition of the database D', we obtain: $\theta_1 S_k(\overline{z}_j, \overline{t}_{jk}) \in S_k(D')$, $1 \leq k \leq n$, $1 \leq j \leq s$, $\theta_1 S_{n+e}(\overline{z}_j, \overline{t}_{jn+e}) \notin S_{n+e}(D')$, $1 \leq e \leq p$, $1 \leq j \leq s$. These statements imply $\theta_1 f(\overline{z}_j) \in f(D')$, $1 \leq j \leq s$. Thus, using the hypothesis $I \subseteq V(\tau_\theta(OldD))$, we get $I \subseteq V(\tau_{\theta_1}(T))$, i.e. $I \subseteq V(D')$. The second part of the statement (v) results from the definition of D'. □

Theorem 1. *Let D be a database defined on Dom such that $I \subseteq V(D)$, and $Rel(D) \subseteq EDB(\mathcal{V})$. Then, there exist a terminal node α in $TREE(I, \mathcal{V})$*

having $TERM(\alpha) = 3$*, a substitution* θ *from* $OldCV(\alpha)$ *into Dom, an injective mapping* τ_θ *defined on* $Class_{\pi(\alpha)}$ *and having its values in Dom such that the following assertions are true:*

(a) $I \subseteq V(D')$*, where* $D' = \tau_\theta(OldD(\alpha))$.
(b) *For each atom* $R(\overline{w})$ *from* $D - D'$*, the vector* $\overline{w}$ *has at least a component that does not belong to* $V' = \theta(NewCV(\alpha))$.

Proof. Firstly, we assign the elements to the root α_0 of $TREE(I, \mathcal{V})$. We take $OldCV = C$, the set of all constants from I, θ the unit substitution: $\theta(c) = c$, for each $c \in C$. It results that the partition π corresponding to θ (Lemma 1) is the discrete partition, $OldD = \{\eta_\pi V(\overline{w}_1), ..., \eta_\pi V(\overline{w}_m)\}$. Let $f = V$ and $I' = OldD$. The substitution θ satisfies: $\theta V(\overline{w}_i) \in V(D)$, because $I \subseteq V(D)$. Since $\tau_\theta(OldD) = I$, we have $I \subseteq V(\tau_\theta(OldD))$. Thus, we can use Proposition 3, because the necessary conditions are satisfied. Applying Proposition 3 for the node α_0, we pass from the node α_0 to a successor β of α_0 from $\text{TREE}(I, \mathcal{V})$, and for the node β the hypothesis of Proposition 3 are again satisfied. Let $\alpha_0, \alpha_1, ..., \alpha_q = \alpha$ be the path from the root α_0 to α. Let $\theta(\alpha_i)$ be the substitution computed for the node α_i, using Proposition 3. Since $\theta(\alpha_{j+1})$ is an extension of $\theta(\alpha_j)$, $0 \leq j < q$, and the statements (a) and (b) from Proposition 3 are true for each node α_j, $0 \leq j \leq q$, we obtain that $TERM(\alpha) = 3$. □

Theorem 2. *Let* α *be a node terminal from* $TREE(I, \mathcal{V})$ *having* $TERM(\alpha) = 3$*. Let* τ *be an injective mapping from* $Class_{\pi(\alpha)}$ *into Dom. Let* D' *be the database defined on Dom such that* $D' = \tau(OldD(\alpha))$*. Then we have:* $I \subseteq \mathcal{V}(D')$.

Proposition 4. *Let* $NewCV$ *be a set of constants and variables and* π *a partition defined on* $NewCV$*. Let* T *be a database defined on* $Class_\pi$*, and* τ *an injective mapping from* $Class_\pi$ *into Dom. Then we have:* $\tau(Q(T)) = Q(\tau(T))$*, for each query* Q *expressed as non-recursive datalog program and having* $EDB(Q) \subseteq EDB(\mathcal{V})$.

5 Computing of Certain Answers

In this section, we give some results necessary to compute all certain answers corresponding to I, $\mathcal{V}$ and Q. Firstly, we need to give a definition that points out a class of tuples, called $C - tuples$.

Definition 6. *Let* $OldCV$ *be a set of constants and variables,* π *a partition defined on* $OldCV$*, and* $\overline{w}$ *a tuple on* $Class_\pi$*. The tuple* $\overline{w}$ *is called a* $C - tuple$*, if each of its components contains a constant.*

Proposition 5. *Let* α *be a terminal node from* $TREE(I, \mathcal{V})$*, with* $TERM(\alpha) = 3$*,* $\pi(\alpha)$ *the partition corresponding to* α*,* $T = OldD(\alpha)$ *the database associated to* α*, with elements from* $Class_{\pi(\alpha)}$*. Let* $T_a = Q(T)$ *be the answer of* Q *for* T *and a tuple* $\overline{w}_a$ *from* T_a.

(a) If the tuple $\overline{w}_a$ is not a $C-tuple$, then for each injective mapping τ from $Class_{\pi(\alpha)}$ into Dom, that preserves the constants, the tuple $\tau(\overline{w}_a)$ is not a certain answer under OWA for I, $\mathcal{V}$ and Q.
(b) If all tuples from T_a are not $C-tuples$, then there are not certain answers under OWA.

We remark that for a $C-tuple$ $\overline{w}_a$ from T_a, there exists only one injective mapping that preserves constants, denoted τ_0, and defined by: $\tau_0(t) = c$ if $c \in [t]_{\pi(\alpha)}$. Moreover, the tuple $\tau_0(\overline{w}_a)$ is a candidate for the set of certain answers.

We can specify a procedure called $CertAnsO$ that computes the set of certain answers. To compute certain answers under CWA, we consider only the terminal nodes α from $TREE(I, \mathcal{V})$ having $TERM(\alpha) = 3$ and satisfying the restrictions: $V(T)$ only contains $C-tuples$ and $\tau_0 V(T) = I$, where $T = OldD(\alpha)$. Thus one can obtain easily a procedure, that computes the certain answers under CWA.

6 Conclusion

In this paper, we have presented a method to compute the certain answers in case when views are expressed as non-recursive datalog programs with negation and queries as non-recursive semi-positive datalog programs with negation. For the future work, it is interesting to analyze whether our method can be extended in cases when views and queries are expressed by other types of datalog programs.

References

1. Abiteboul, S., Duschka, O.M.: Complexity of answering queries using materialized views. In: PODS, pp. 254–263 (1998)
2. Afrati, F., Li, C., Mitra, P.: Rewriting queries using views in the presence of arithmetic comparisons. Theoretical Computer Science 368, 88–123 (2006)
3. Arenas, M., Bertossi, L., Chomicki, J.: Answer set for consistent query answering in inconsistent databases. Theory and Practice of Logic Programming 3, 394–424 (2003)
4. Call, A., Lembro, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: IJCAI 2003, pp. 16–21 (2003)
5. Call, A., Lembro, D., Rosati, R.: On the Decodability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In: PODS, pp. 260–271 (2003)
6. Calvanese, D., Giacomo, G. De, Lenzerini, M., Vardi, M.Y.: View-based query processing. On the relationship between rewriting, answering and losslessness. Theoretical Computer Science 371, 169–182 (2007)
7. Calvanese, D., Giacomo, G. De, Lenzerini, M., Vardi, M.Y.: Answering Regular Path Queries Views. In: Proc. of the 16th IEEE Int. Conf. on Data Engineering, ICDE, pp. 389–398 (2000)
8. Caniupan, M., Bertossi, L.: The consistency extractor system: Answer set programs for consistent query answering in databases. Data and Knowledge Engineering 69, 545–572 (2010)

9. Chomicki, J.: Consistent Query Answering, Recent Developments and Future Directions., http://cse.buffalo.edu/~chomicki/papers-iicis0
10. Duschka, O.M., Genesereth, M.R., Levy, A.: Recursive Query Plans for Data Integration. Journal of Logic Programming 43(1), 49–73 (2000)
11. Fagin, R., Kolaitis, P.G., Popa, L., Miller, R.J.: Data Exchange: Semantics and Query Answering. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 207–224. Springer, Heidelberg (2002)
12. Flesca, S., Greco, S.: Rewriting queries using views. IEEE Trans. Knowledge Data Engineering 13(6), 980–995 (2001)
13. Millstein, T., Halevy, A., Friedman, M.: Query containment for data integration systems. JCSS 66, 20–39 (2003)
14. Nash, A., Segoufin, L., Vianu, V.: Views and Queries. Determinacy and Rewriting. ACM TODS 35(3) (2010)