Algorithm for Term Linearizations of Aggregate Queries with Comparisons

Victor Felea[1] and Violeta Felea[2]

[1] CS Department, Al.I. Cuza University, Iasi, Romania
felea@infoiasi.ro
[2] FEMTO-ST, Franche-Comté University, Besançon, France
violeta.felea@femto-st.fr

Abstract. We consider the problem of rewriting queries based exclusively on views. Both queries and views can contain aggregate functions and include arithmetic comparisons. To study the equivalence of a query with its rewriting query, the so called "linearizations of a query" need to be computed. To find the linearizations of a query, the linearizations of terms from the query need to be generated. We propose an algorithm to find these term linearizations and give a bound for its time-complexity.

1 Introduction

The problem of rewriting queries using views (RQV) is expressed, informally, as follows: given Q a query over a database schema, and $\mathcal{V}$ a set of view definitions, is it possible to answer the query Q using only the answers to the views from $\mathcal{V}$? One of the application fields of RQV concerns extracting and gathering information on the environment, based on sensor networks. Small inexpensive devices, namely sensors, communicate information to collecting points, called sinks or base stations. In this context, of information collection out of several sources (stored in views), located on sensor nodes, expressing queries using views. In [7], the authors apply the notions of query interface for data integration to sensor networks.

In case of finding a query Q', corresponding to a given query Q, and expressed by the views from $\mathcal{V}$, such that Q is equivalent to Q', the problem of (RQV) for the query Q has a solution. This query Q' is called a rewriting query of Q.

The equivalence problem of two aggregate queries with comparisons was investigated in [4], [1], [2], [5]. The equivalence problem of the two queries is achieved using the equivalence of their reduced queries. A reduced query associated to a query Q uses the so-called "linearizations" (or complete ordering) of the set of all terms from Q.

In [3], query equivalence between view-based query and schema query, using linearizations, is proven; no linearization computation has been given. This is the main aim of this work: to propose an algorithm to compute linearizations. By our best knowledge, an algorithm to obtain all linearizations of terms from a query has not been proposed in the literature.

In the paper, we consider the problem of generating all linearizations corresponding to a query that uses arithmetic comparisons and aggregate functions; we consider a motivated example (see section 2) and give an algorithm that computes these linearizations, and its time complexity in section 3. We finally conclude.

2 Motivating Example

Let us give an example about a schema regarding sensor networks used in statistical applications concerning the weather. The network nodes are organized in clusters [6], one common architecture used in sensor networks.

Example 1. We assume there exist sensors for the following type of data: humidity, mist and temperature. One sensor can retrieve unique or multiple environmental data, e.g. temperature and humidity, or only temperature values. Sensors are placed in several locations, and have an identifier, a type, and they are grouped by regions. The type of a sensor characterizes the set of data types that it senses. A location is identified by two coordinates *lat*, *long*, where *lat* is the latitude and *long* is the longitude. A region has an identifier and a unique name. Let us consider the relational schema $\mathcal{S}$ consisting of $\{REG, POINT, SENSOR, DATA\}$, where *REG* represents regions and has *idReg* as region identifier, and *regName* as region name; *POINT* represents the locations from the regions, and uses the *idReg* attribute as the region identifier, and the *lat*, *long* attributes as the coordinates of a location from the region identified by *idReg*; *SENSOR* represents information about sensors with *idS* the sensor identifier, *lat*, *long* - the sensor coordinates, *typeS* its type related to the sensing capacity (e.g. *typeS* equals 1 for temperature and humidity, 2 for temperature and mist, 3 for humidity and mist, and so on), and *chS* designs if the sensor is a cluster head (its value equals 1 or 2). We assume here that the type of the sensors that can sense temperature equals either 1 or 2; *DATA* represents the values registered by sensors, and has the following attributes: *idS* - the sensor identifier, *year*, *month*, *day*, *period* - the date of the sensed information, the daily frequency of sensed data, expressed in seconds, and *temp*, *humid*, *mist* - the temperature, humidity, and mist values, respectively.

Let us consider the following views:

V_1: Compute all pairs of the form (*idReg*, *regName*), where *idReg* is the region identifier, and *regName* is the name of the region identified by *idReg*, such that in the region *idReg* at least a cluster head exists.
V_2: Compute the maximum temperature for each region, each network location, and for every year.

Let us consider the following queries:

Q_1: Find the maximum of temperature values for the *"NORTH"* region, between 2003 and 2010, considering data sensed only by cluster head sensors.
Q_2: Find the maximum temperature corresponding to the *"NORTH"* region, on September 22nd, 2009.

Example 2. Let us consider the schema, queries, and views from Example 1. We denote the attributes *idReg*, *regName* from *REG* by x, y_1, respectively, *idReg*, *lat*, *long* from *POINT* by y_2, y_3, y_4, and *idS*, *lat*, *long*, *typeS*, *chS* from *SENSOR* by z_1, z_2, z_3, t_1, t_2, and the attributes from the schema *DATA* by $z_5, z_6, z_7, z_8, z_9, t, z_{11}$, z_{12}. To be short, we denote by D, R, S, P the relational symbols *DATA*, *REG*, *SENSOR*, *POINT*, respectively. Let c_1 =*"NORTH"*, $c_2 = 2003$ and $c_3 = 2010$. To obtain a condition associated to Q_1, we replace the variables y_1, t_2 by c_1, 1, respectively. Since $y_2 = x$, $z_2 = y_3$, $z_3 = y_4$, and $z_5 = z_1$, the query Q_1 is expressed as the following condition:
$Q_1 : h(x, max(t)) \leftarrow D(z_1, z_6, z_7, z_8, z_9, t, z_{11}, z_{12}) \wedge R(x, c_1) \wedge S(z_1, y_3, y_4, t_1, 1) \wedge P(x, y_3, y_4) \wedge C$, where $C \equiv ((t_1 = 1) \vee (t_1 = 2)) \wedge (z_6 \geq c_2) \wedge (z_6 \leq c_3)$.

The query Q_2, and the views V_1, V_2 defined in Example 1, have similar expressions.

3 Term Linearizations of Aggregate Queries

In this section we specify some notions about the *term linearizations* and give an algorithm to compute these *linearizations* constructed by the terms from the conditions contained in the aggregate query. For other notions, see [3].

Two aggregate queries Q_1 and Q_2 having identical heads are said *equivalent*, if their answers are equal, i.e. $Q_1(D) = Q_2(D)$, for any database D defined on *Dom*. Let f_i be the body of Q_i. Since each f_i is a disjunction of relational atoms and of an expression constructed from comparison atoms using the conjunction and the disjunction operators, it can contain comparisons of terms. The comparisons of an expression f_i induce, in general, a partial order among the constants and variables of the query. To study the equivalence of two queries, we need to consider linear orderings such that a given partial ordering of the set of terms is compatible with a set of linear orderings of the same set of terms.

The set of all terms from a query can be considered as a set of groups, where all attributes of a group correspond to the same value domain. Let C be a set of comparison atoms, and $t_1 < t_2$ be a comparison atom. We say that C implies $t_1 < t_2$, denoted $C \models t_1 < t_2$, if for each substitution τ, the statement ($\tau\delta$= *true* for each $\delta \in C$) implies $\tau t_1 < \tau t_2$. For a comparison atom having the form $t_1 = t_2$ or $t_1 > t_2$ the implication relation of a comparison atom from a set of comparison atoms is similar. Let T be a set of terms (variables or constants) that are semantically equivalent. A *linearization* of T is a set of comparisons L having the terms in T such that for any two different elements t_1, t_2 from T, exactly one of the following comparisons: $t_1 < t_2$, $t_2 < t_1$, or $t_1 = t_2$ is implied by L. If T has $t_1, \ldots, t_p$ as elements, let $(t'_1, \ldots, t'_p)$ be a permutation of the elements t_j, $1 \leq j \leq p$. Then we can represent a linearization L of T as $t'_1\rho_1t'_2\rho_2 \ldots t'_{p-1}\rho_{p-1}t'_p$, where the operators ρ_i are '$<$' or '$=$'. We say that a substitution τ from $\{t'_1, \ldots, t'_p\}$ into *Dom*, that preserves constants, satisfies the linearization L, denoted $\tau(L) = true$, if for each $i, 1 \leq i < p$, $\tau t'_i = \tau t'_{i+1}$, when ρ_i is '$=$', and $\tau t'_i < \tau t'_{i+1}$, when ρ_i is '$<$'. We denote by $t_1 = t_2 = \ldots = t_h$, the set of comparisons $t_i = t_{i+1}$ contained in L, for any $i, 1 \leq i < h$. Intuitively, a linearization is a list of different variables and constants such that between two elements of the list there exists one of the '=' or '<' operators. In this way, a linearization produces a partition of all terms into *equivalence classes*; an equivalence class contains all terms $t_1, \ldots, t_h$ such that $t_1 = t_2 = \ldots = t_h$.

We say that a set of comparisons C is *satisfiable*, if there exists a substitution τ from terms into Dom such that τ satisfies all comparisons from C. We say that τ satisfies C or $\tau(C) = true$, if $\tau t_1 < \tau t_2$ if $C \equiv t_1 < t_2$, and $\tau t_1 = \tau t_2$ if $C \equiv t_1 = t_2$. The value of $\tau(C)$ is extended naturally to expressions C obtained from basic comparisons using the conjunction and disjunction operators. We say that a linearization L of T, and a set of comparisons C are compatible, if $L \cup C$ is satisfiable.

In the following we give a method to obtain all linearizations of T, compatible with C. Let C be the conjunction of the comparisons $t_i \sigma_i t'_i$, i.e. $C = (t_1 \sigma_1 t'_1) \wedge \ldots \wedge (t_h \sigma_h t'_h)$, where the operators σ_i are relational operators. Let C' be the formula obtained from C, by replacing $(t_i \leq t_j)$ with $(t_i < t_j) \vee (t_i = t_j)$, and $(t_i \geq t_j)$ with $(t_j < t_i) \vee (t_i = t_j)$. Using for the expression C', the distributivity of the conjunction versus the disjunction, we obtain a formula C'' having the form: $C'' = E_1 \vee E_2 \vee \ldots \vee E_p$ (1), where E_j is a conjunction of comparisons of the form $t_i < t_j$ or $t_i = t_j$. In this disjunction, we take only consistent conjunctions by eliminating those conjunctions that are inconsistent, i.e. which contain at least two comparisons of the form: $t_1 < t_2$ and $t_1 = t_2$, or two comparisons of the form $t_1 < t_2$ and $t_2 < t_1$. Let E be an arbitrary conjunction from C''. Associated to E, we construct a forest, denoted G_E. Firstly, for the set of all comparisons S_E of the form $t_i = t_j$ from E, we take the transitive and symmetrical closure of S_E. This closure produces a set of equivalence classes $Cl_1, \ldots, Cl_q$. Let us denote by $Term1 = \{t_1, \ldots, t_p\}$, the remainder of the variables and the constants from E. The forest G_E associated to E is defined as follows: its nodes are labeled with Cl_j, $1 \leq j \leq q$ and $t_i, 1 \leq i \leq p$. Let $Var(C)$ be the set of variables from C and $Const(C)$ the set of constants from C.

The edge set $\mathcal{N}$ of the forest G_E is specified as follows:

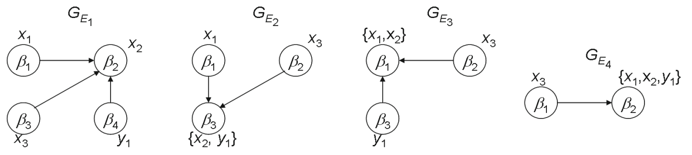
- Let t_1 and t_2 be variables or constants. If $t_1 < t_2$ appears in E, and t_1 and t_2 occur in $Term1$, then $(t_1, t_2) \in \mathcal{N}$.
- Let Cl_j be a class and t_2 from $Term1$. If there exists $t' \in Cl_j$ such that $t' < t_2$ occurs in E, then $(Cl_j, t_2) \in \mathcal{N}$.
- Let t_1 be from $Term1$, and Cl_h a class. If there exists $t' \in Cl_h$ such that $t_1 < t'$ occurs in E, then $(t_1, Cl_h) \in \mathcal{N}$.
- Let Cl_j, Cl_h be two different classes. If there exist the terms t_1, t_2, where $t_1 \in Cl_j$ and $t_2 \in Cl_h$ such that $t_1 < t_2$ appears in E, then $(Cl_j, Cl_h) \in \mathcal{N}$.

Since the labels associated to different nodes contain disjoint sets of terms, we identify the nodes with their labels. If a class Cl_j consists of the elements $t_1, \ldots,$ t_s, we denote it as $\{t_1, \ldots, t_s\}$ or $\{t_1 = \ldots = t_s\}$.

Example 3. Let $C \equiv (x_1 \leq x_2) \wedge (x_2 > x_3) \wedge (y_1 \leq x_2)$. The formula corresponding to C has the form: $C' = [(x_1 < x_2) \vee ((x_1 = x_2)] \wedge (x_3 < x_2) \wedge [(y_1 < x_2) \vee (y_1 = x_2)]$. The expression C'' has the form: $C'' = E_1 \vee E_2 \vee E_3 \vee E_4$, where

$E_1 = (x_1 < x_2) \wedge (x_3 < x_2) \wedge (y_1 < x_2)$, $E_2 = (x_1 < x_2) \wedge (x_3 < x_2) \wedge (y_1 = x_2)$,
$E_3 = (x_1 = x_2) \wedge (x_3 < x_2) \wedge (y_1 < x_2)$, $E_4 = (x_1 = x_2) \wedge (x_3 < x_2) \wedge (y_1 = x_2)$.

The forests associated to each of the four expressions are given in the following figure, where nodes are labeled β_i.



3.1 A Method to Obtain the Linearizations of *C*

In this subsection, we specify a method to obtain all linearizations of $Var(C) \cup Const(C)$ compatible with C and corresponding to a forest G_E associated to an expression E from C''. The construction of this set of linearizations will be done recursively in function of the number of edges from G_E.

- The base step: Let G_E having zero edges, and $\beta_1, \ldots, \beta_p$ its nodes. Let $i_1, \ldots, i_p$ be a permutation of the indexes $1, 2, \ldots, p$. Let us denote by $\mathcal{M}^L_{G_E}$ the set of linearizations defined by G_E. For this base step, we take:
  $\mathcal{M}^L_{G_E} = \{\beta_{i_1}\sigma_1\beta_{i_2}\sigma_2 \ldots \beta_{i_{p-1}}\sigma_{p-1}\beta_{i_p} | (i_1, \ldots, i_p)$ is a permutation of $\{1, 2, \ldots, p\}, \sigma_i \in \{'<', '='\}\}$.
  We make the following convention: $\{t_1 = t_2 = \ldots = t_s\} = \{t'_1 = t'_2 = \ldots = t'_r\}$ becomes: $\{t_1{=}t_2{=}\ldots{=}t_s{=}t'_1{=}t'_2{=}\ldots{=}t'_r\}$.
- The inductive step: Assume that we have computed the set of linearizations corresponding to any forest G having at most n edges, where n is a natural number. Let G be a forest having $n{+}1$ edges. Let t_0 be an initial node from G, and $\gamma_1, \ldots, \gamma_k$ the immediate successors of t_0. Let G' be the forest obtained from G by deleting the edges (t_0, γ_j), $1 \leq j \leq k$, and deleting the node t_0. Since G' has at most n edges, by induction we have computed $\mathcal{M}^L_{G'}$. Using the linearizations of G', we compute the linearizations for G. Let $L \equiv s_1 < s_2 < \ldots < s_m$ be an element of $\mathcal{M}^L_{G'}$, where m is the number of the nodes of G'. An element s_i contains a set of terms from $Var(C) \cup Const(C)$. The label γ_j belongs to a unique s_i, for each $j, 1 \leq j \leq k$ ($\gamma_j \subseteq s_i$). Let us denote by $ind(j)$ this natural number i. Let i_0 be the minimum of $ind(j)$, for each $j, 1 \leq j \leq k$. Now, we define a set of linearizations for G corresponding to L by inserting t_0 in L before s_{i_0}. Let us denote by $Insert(t_0, L, l)$ the linearizations obtained from L by inserting t_0 between the positions l and $l + 1$. There are two linearizations in $Insert(t_0, L, l)$ in case $l \geq 1$, denoted L_1, L_2, where $L_1 \equiv s_1 < s_2 < \ldots < s_l < t_0 < s_{l+1} < \ldots < s_m$, and $L_2 \equiv s_1 < s_2 < \ldots < \{s_l = t_0\} < s_{l+1} < \ldots < s_m$. If $l = 0$ the insertion takes place before s_1 and $Insert(t_0, L, l)$ consists of one sequence, namely L_1, where $L_1 \equiv t_0 < s_1 < s_2 < \ldots < s_m$. For the linearization L and the node t_0, we consider the union of the sets $Insert(t_0, L, l)$, for each $l, 0 \leq l < i_0$. Let us denote this set by $\mathcal{M}(t_0, L)$.

Now, we define the set of linearizations corresponding to G as follows: $\mathcal{M}_G^L = \{L' | (\exists L)(L \in \mathcal{M}_{G'}^L)$ such that $L' \in \mathcal{M}(t_0, L)\}$.

Let E_j be an expression from relation (1), $1 \leq j \leq p$. Let G_{E_j} be the forest corresponding to E_j. Let $\mathcal{M}_{G_{E_j}}^L$ be the set of linearizations computed for the forest G_{E_j}. Let $\mathcal{M}(C)$ be the union of the $\mathcal{M}_{G_{E_j}}^L$ sets, for each $j, 1 \leq j \leq p$. Regarding these notations, we have the following results.

Lemma 1. *Let E be a consistent conjunction of comparisons of the form $t_i < t_j$ or $t_i = t_j$. Let L be a linearization compatible with E, and G_E the forest associated to E. Then, we have $L \in \mathcal{M}_{G_E}^L$.*

Proof. Using the induction on the number of the comparisons of the form '<' from E (this number is equal to the number of the edges from G_E).

Theorem 1. *Let L be a linearization of $Var(C) \cup Const(C)$ and $\mathcal{M}(C)$ the set of linearizations computed for C as above. Then, we have: L is compatible with C iff $L \in \mathcal{M}(C)$.*

Proof. Let L be a linearization from $\mathcal{M}(C)$. There exists an integer $j, 1 \leq j \leq p$ such that $L \in \mathcal{M}_{G_{E_j}}^L$. The linearization L can be represented as $Cl(t_1) < Cl(t_2) < \ldots < Cl(t_m)$, where $Cl(t_i)$ are equivalence classes corresponding to L, and t_i is an element from that class. A class $Cl(t_i)$ consists of a single variable, a single constant, or a set of terms $\{t'_1, \ldots, t'_h\}$, $h \geq 2$, and this set contains at most one constant. The union of the classes $Cl(t_i)$ is $Var(C) \cup Const(C)$. Let us define the substitution τ as follows: if $Cl(t_i) = \{t'_1, \ldots, t'_h\}$, then $\tau t'_j = \tau t'_l$, $1 \leq j < l \leq h$. The substitution τ is extended to classes: $\tau Cl(t_j) = \tau t_j$. For two classes, we take the values for τ such that if $Cl(t_i) < Cl(t_j)$, then $\tau t_i < \tau t_j$. In this manner, we have $\tau(L)$ is true. Using the method to construct $\mathcal{M}_{G_{E_j}}^L$, we obtain $\tau(E_j)$ is true, hence $\tau(C)$ is true. That means $L \cup C$ is satisfiable.

Conversely, let L be a linearization compatible with C. It results that there exists an integer $j, 1 \leq j \leq p$ such that L is compatible with E_j. Using Lemma 1, we obtain $L \in \mathcal{M}_{G_{E_j}}^L$, hence $L \in \mathcal{M}(C)$.

Using Theorem 1, we obtain an algorithm that computes the set of all linearizations for a conjunction of comparisons.

3.2 Algorithm to Compute Linearizations of $\mathcal{M}(C)$

Firstly, let us denote by *LinearizationsComp*(G,$\mathcal{L}$) an algorithm that computes the set $\mathcal{L}$ of all linearizations corresponding to the forest G. Let us consider some notations used in this algorithm. We denote by h the number of levels from G. The terminal nodes of G are considered on level 1. Let N_j be the set of all nodes on level j, $1 \leq j \leq h$. We denote by $\mathcal{L}_j$ the set of all linearizations for nodes situated on the levels l in G, where $l \leq j$. We use the notation $\mathcal{M}(\gamma, L)$ for a set of linearizations obtained by inserting the node γ into a linearization L, as specified in subsection 3.1. By $\mathcal{L}_{j,l}$ we denote a set of linearizations of nodes from G, that is a subset of $\mathcal{L}_j$. The algorithm proceeds level by level, beginning with the first level.

```
Algorithm *LinearizationsComp* (input $G$, output $\mathcal{L}$)
compute $h$ the number of levels from $G$
**for all** $j$=1 **to** $h$ **do** compute $N_j$, the set of all nodes on level $j$ **endfor**
compute $\mathcal{L}_1$ the set of linearizations for $N_1$ (see base step from Subsection 3.1)
**if** $h$=1 **then** $\mathcal{L} = \mathcal{L}_1$ exit **endif**
**for all** $j$=2 **to** $h$ **do**
  $\mathcal{L}_j = \emptyset$
  let $N_j = \{\gamma_1, \ldots, \gamma_k\}$
  $\mathcal{L}_{j,0} = \mathcal{L}_{j-1}$
  **for all** $l$=1 **to** $k$ **do**
    $\mathcal{L}_{j,l} = \emptyset$          let $\mathcal{L}_{j,l-1} = \{L_1, \ldots, L_p\}$
    **for all** $s = 1$ **to** $p$ **do**
      compute $\mathcal{M}(\gamma_l, L_s)$ (see Subsection 3.1)
      $\mathcal{L}_{j,l} = \mathcal{L}_{j,l} \cup \mathcal{M}(\gamma_l, L_s)$
    **endfor**
  **endfor**
  $\mathcal{L}_j = \mathcal{L}_{j,h}$
**endfor**
$\mathcal{L} = \mathcal{L}_h$
```

Secondly, we specify the computing of the linearizations corresponding to the expression C that has comparison atoms as base expressions, and is formed of base atoms using the conjunction or disjunction operators. The expression C is equivalent to a disjunction of conjunctions, i.e. $E \equiv E_1 \vee \ldots \vee E_p$. For each expression E_j, we construct the forest G_{E_j}, and call the algorithm *LinearizationsComp* with the parameters G_{E_j} and $\mathcal{L}_j$. The set of the linearizations for C, denoted $\mathcal{M}(C)$, is the union of all $\mathcal{L}_j$, i.e. $\mathcal{M}(C) = \cup_{j=1}^{p} \mathcal{L}_j$.

Example 4. Let us consider Example 2. Let C be the comparison expression from Q_1. Since $z_6 \geq c2$ is equivalent to $(z_6 > c2) \vee (z_6 = c_2)$, and $z_6 \leq c_3$ is equivalent to $(z_6 < c_3) \vee (z_6 = c_3)$, the expression C is equivalent to $C'' \equiv \bigvee_{i=1}^{3} [(t_1 = 1) \wedge E_i]$ $\bigvee_{i=1}^{3} [(t_1 = 2) \wedge E_i]$, where $E_1 \equiv (c_2 < z_6) \wedge (z_6 < c_3)$, $E_2 \equiv (c_2 < z_6) \wedge (z_6 = c_3)$, $E_3 \equiv (z_6 = c_2) \wedge (z_6 < c_3)$. The expression $(z_6 = c_2) \wedge (z_6 = c_3)$ is non satisfiable because all constants are considered different. Concerning the constants c_2 and c_3, we assume $c_2 < c_3$, otherwise the expression C is not satisfiable. As we specified in Section 3, we represent a linearization as a set of groups, where a group is a linearization of all terms corresponding to the same value domain. For the group $\{z_6, c_2, c_3\}$, and the first conjunction from C'', we obtain one linearization: $c_2 < z_6 < c_3$. Hence, to the first conjunction of C'', we have two groups denoted g_1, g_2, where $g_1 \equiv (t_1 = 1)$, $g_2 \equiv c_2 < z_6 < c_3$. We consider an order of linearizations as follows: L_1 for the first conjunction of C'', L_2 for the fourth, L_3 for the second, L_4 for the fifth, L_5 for the third, and L_6 for the sixth. Thus, the linearizations L_i have the following forms:

$L_1 \equiv (t_1 = 1,\, c_2 < z_6 < c_3)$, $L_2 \equiv (t_1 = 2,\, c_2 < z_6 < c_3)$,
$L_3 \equiv (t_1 = 1,\, z_6 = c_3)$, $L_4 \equiv (t_1 = 2,\, z_6 = c_3)$,
$L_5 \equiv (t_1 = 1,\, z_6 = c_2)$ and $L_6 \equiv (t_1 = 2,\, z_6 = c_2)$.

3.3 Complexity Issues

Let us compute the time complexity of algorithm 1. Let G be a forest, with n nodes and p edges and h its number of levels (corresponding to the depth of the forest) with $h \leq p + 1$. For every level j, $1 \leq j \leq h$, let N_j be the number of nodes on level j. We have $\sum_{j=1}^{h} N_j = n$. The set L_j from the algorithm gives all the linearizations obtained using nodes from levels 1 to j. The number of terms for each linearization from L_j is $length_j = \sum_{k=1}^{j} N_k$. Let us denote by $|L_j|$ the cardinal of the L_j set. We have $|L_1| = N_1!$ because there is no edge between the nodes on the first level. For the other levels, we have: $|L_{j+1}| \leq N_{j+1} * length_j * |L_j|$, $1 \leq j \leq h - 1$.

We obtain $|L_h| \leq N_h * N_{h-1} * \ldots * N_1 * (N_{h-1} + \ldots + N_1) * \ldots * (N_2 + N_1) * N_1!$.

The set of terms can be grouped in classes, a class contains all terms having the same value domain. If m is the maximum number of elements in these classes, then m is the number of the forest nodes in the given algorithm.

4 Conclusion

In the paper, we propose an algorithm to compute linearizations and evaluate its complexity. These are useful in the process of constructing and proving the query equivalence between query expressed on the database schema and its rewritings using views. Queries and views considered here contain aggregate functions and arithmetic comparisons. Future work will focus on the aspects concerning using linearizations in the problem of queries equivalence when queries and views contain negations.

References

1. Afrati, F., Li, C., Mitra, P.: Rewriting queries using views in the presence of arithmetic comparisons. Theoretical Computer Science 368, 88–123 (2006)
2. Cohen, S.: Containment of aggregate queries. ACM SIGMOD 34(1), 77–85 (2005)
3. Cohen, S., Nutt, W., Serebrenik, A.: Rewriting aggregate queries using views. In: PODS, pp. 155–166 (1999)
4. Cohen, S., Nutt, W., Sagiv, Y.: Deciding equivalences among conjunctive aggregate queries. Journal of the ACM 54(2), 1–50 (2007)
5. Grumbach, S., Rafanelli, M., Shurin, S.: On the equivalence and rewriting of aggregate queries. Acta Informatica 4(8) (2004)
6. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-efficient Communication Protocol for Wireless Microsensor Networks. In: 33rd Annual Hawaii International Conference on System Sciences (HICSS-33), pp. 3005–3014 (2000)
7. Madden, S., Szewczyk, R., Franklin, M., Culler, D.: Supporting aggregate queries over ad-hoc wireless sensor networks. In: Proc. of 4th IEEE Workshop on Mobile Computing Systems and Applications, pp. 49–58 (2002)