

Abstract objects to represent large answers to queries in a concise form

Robert Demolombe

ONERA Toulouse, 2 Avenue E. Belin BP 4025, 31055 Toulouse, France

Abstract. Abstract objects can be used to represent in a concise form answers that are communicated by telephone. We present a formal framework in first order logic in which are defined abstract answers, their lower bounds and their upper bounds. Algebraic formulas are given to efficiently compute the abstract answers from the abstraction of the predicates that occur in a given query. We also present a method to reduce the error caused by the abstraction process that has to be computed when users want to get an exact answer.

Keywords: Answer generation, Knowledge representation.

1 Introduction

In the future there will be a lot of applications where people ask queries to databases by telephone. In these contexts, if answers to queries are made of a large number of tuples, users cannot keep in mind the whole answer. Even for answers that contain around 10 tuples, it is likely that when the user hear the last tuple he has forgotten the first one.

A possible solution to this problem is to “synthesize” the answer by using abstract objects to represent sets of tuples, in order to get more concise answers. This approach has been previously investigated by other authors, in particular by T. Ellman in [Ell95] and T. Imielinski in [Imi87]. Before we present our proposed solution in formal terms we shall give an intuitive presentation with an example.

Let's consider, for instance, a database that contains statistical information about weather in the main cities in France. This information is formally represented with the predicate $weather(x, y, z)$, where x , y and z respectively denote a city, a day in the year, and the weather. For instance, the fact that it is usually cold in Strasbourg on January 20th is represented by: $weather(Strasbourg, 20/01, cold)$. Queries are expressed by formulas of a first order predicate calculus language. For instance the query: *in which cities the weather is the same as in Nice on December 25th?*, is expressed by: $\exists y(weather(Nice, 25/12, y) \wedge weather(x, 25/12, y))$, and the query: *in which cities and on which days the weather is sunny and not warm?*, is expressed by: $weather(x, y, sunny) \wedge \neg weather(x, y, warm)$. The answer to the second query may be, for instance, the relation R.

* This work has been partially supported by CNET (French Telecom) under the contract number 961B317.

A more concise answer can be obtained if the elements in the definition domains are represented by abstract objects. For instance, days can be abstracted into months, and months can be abstracted into seasons. In the same way, cities can be abstracted into “counties”, and “counties” can be abstracted into “regions”. The definitions of these abstract objects are, for instance:

January={01/01,02/01,...,31/01}, ..., December={01/12,,02/12,...,31/12},
 Winter={January,February,March}, ..., Autumn={September,October,...}.
 Bas-Rhin={Strasbourg,Saverne,...}, Haut-Rhin={Colmar,Mulhouse,...},
 Moselle={Metz,Thionville,...}, ... Alsace={Bas-Rhin, Haut-Rhin,Moselle,...},
 Aquitaine={Gironde,Landes,...},

Then, the same answer can be represented at the first abstraction level by the relation R1, and at the second abstraction level by the relation R2.

R					
Strasbourg	01/01	R1			
Strasbourg	02/01	Bas-Rhin	January		
Colmar	01/01	Bas-Rhin	February	R2	
Colmar	03/01	Haut-Rhin	January	Alsace	Winter
Metz	02/01	Moselle	January
Thionville	01/01		
...	...				

If the intuitive meaning of the tuple <Alsace,Winter> in R2 is that it is cold in Alsace in winter, it may be that the weather is not cold in all the cities in Alsace on Spring, nor in all the cities in Bas-Rhin on February. So, we have to make clear what is the meaning of the presence or of the absence of a tuple in these abstract relations.

We have considered here two different possible meanings. For instance, the meaning of the tuple <Bas-Rhin,February> can be that for every city x in Bas-Rhin, and for every day y on February, the weather is cold in x on y , or it can be that there exists at least one city x in Bas-Rhin and one day y on February such that the weather is cold in x on y .

If we formally see abstract objects (at any abstract level) as unary predicates that takes their values in the domain of the database relations, these two definitions can be expressed by:

$\forall x \forall y (Bas-Rhin(x) \wedge February(y) \rightarrow weather(x, y, cold))$, and
 $\exists x \exists y (Bas-Rhin(x) \wedge February(y) \wedge weather(x, y, cold))$.

The fact that Strasbourg is abstracted into Bas-Rhin, that Bas-Rhin is abstracted into Alsace, and that Alsace is abstracted into France, could be formally represented by: $Bas-Rhin(Strasbourg)$, $Alsace(Bas-Rhin)$ and $France(Alsace)$. However, with this formalisation we would have unary predicates, like $Bas-Rhin(x)$, as an argument of another unary predicate, like in $Alsace(Bas-Rhin)$. That means that we would have an higher order logic. To avoid this complication, and to stay in first order logic, we have preferred to represent these facts by the formulas: $\forall x (Bas-Rhin(x) \rightarrow Alsace(x))$, $\forall x (Alsace(x) \rightarrow France(x))$ and $Bas-Rhin(Strasbourg)$. For the homogeneity, the fact $Bas-Rhin(Strasbourg)$ could equivalently be represented by: $\forall x (x = Strasbourg \rightarrow Bas-Rhin(x))$.

In this paper a formal logical framework is presented to give a precise semantics to abstract answers to queries. In particular we define upper bounds and lower bounds for abstract answers (section 2). Then, we define an extended relational algebra to compute these upper bounds and lower bounds from abstract relations. We also give a method to reduce the error of abstract answers if they are refined at the concrete level (section 3). Finally, some guidelines for the implementation are presented. In particular we give indications of how to restrict the query language to make possible the translation of expressions in the extended relational algebra into the standard relational algebra, which can easily be translated into SQL (section 4).

2 Formal logical framework

Knowledge representation at the level 0.

Let's call L a first order predicate calculus language without function symbols and with the logical connectives: negation (denoted by \neg) and disjunction (denoted by \vee), and the existential quantifier (denoted by \exists). It is assumed that variables symbols are of the form: $x_1, x_2, \dots, x_n, \dots$

Other logical connectives: conjunction (denoted by \wedge), implication (denoted by \rightarrow), and universal quantifier (denoted by \forall), are defined as usual.

A "database" DB is a set of range restricted definite Horn clauses. We denote by CWA the set of clausal world assumption axioms for the database DB , and we denote by CDB the set of sentences $DB \cup CWA$. We denote by D the set of constant symbols that occur in DB ².

Let's call L_0 an extension of the language L with a new unary predicate symbol $d(x_1)$.

Let's call DB_0 the set of sentences $DB_0 = DB \cup (\bigcup_{c \in D} d(c))$. We denote by CDB_0 the set of sentences $CWA_0 \cup DB_0$, where CWA_0 is the set of closed world assumption axioms for DB_0 .

Knowledge representation at the abstraction level l .

The language L_l is defined from the language L_{l-1} by adding to the predicate symbols in L_{l-1} the new unary predicate symbols: a_1^l, \dots, a_n^l .

The domain D_l at the abstraction level l is the set of predicate symbols: a_1^l, \dots, a_n^l .

The database DB_l is defined from the database DB_{l-1} by adding to DB_{l-1} the following formulas:

- $\forall x_1 (d(x_1) \leftrightarrow a_1^l(x_1) \vee \dots \vee a_n^l(x_1))$, where a_1^l, \dots, a_n^l are the elements of D_l
- for every element a_i^l in D_l :
 - a set of formulas of the form: $a_i^l(c_1), \dots, a_i^l(c_{n_i})$, where c_1, \dots, c_{n_i} are in D , and
 - $\exists x_1 a_i^l(x_1)$,

² To avoid too complex notations, we assume here that there is only one domain. It would not be a problem to define a first order many sorted predicate calculus, and to define domain closure axioms for each sort.

- for every pairs of distinct elements a_i^l and a_j^l in D_l : $\neg\exists x_1(a_i^l(x_1) \wedge a_j^l(x_1))$,
- for every element a_j^{l-1} in D_{l-1} : $\forall x_1(a_j^{l-1}(x_1) \rightarrow a_k^l(x_1))$, where a_k^l is some element in D_l .

Let's call CDB_l the set of sentences $CWA_l \cup DB_l$, where CWA_l is the set of closed world assumption axioms for DB_l . It is assumed that every CDB_l is consistent.

From an intuitive point of view, for every l the elements of D_l define a partition on the elements of D . Also, if we denote by $a_k^l(a_j^{l-1})$ the formula $\forall x_1(a_j^{l-1}(x_1) \rightarrow a_k^l(x_1))$, $a_k^l(a_j^{l-1})$ can be interpreted as the fact that a_j^{l-1} is in the extension of a_k^l at the abstraction level $l-1$. Then, we can intuitively say that the elements of D_l define a partition on the elements of D_{l-1} .

We shall use the following notations:

$\mathbf{x} = \langle x_{i_1}, \dots, x_{i_n} \rangle$, where $i_1 < i_2 < \dots < i_n$; $\mathbf{c} = \langle c_1, \dots, c_n \rangle$, where every c_i is in D ; $a^l = \langle a_{i_1}^l, \dots, a_{i_n}^l \rangle$, where every $a_{i_j}^l$ is in D_l ; $a^l(\mathbf{x}) = a_{i_1}^l(x_{i_1}) \wedge \dots \wedge a_{i_n}^l(x_{i_n})$; $\exists \mathbf{x} = \exists x_{i_1} \dots \exists x_{i_n}$; $\forall \mathbf{x} = \forall x_{i_1} \dots \forall x_{i_n}$.

Definition 1. Supremum of f at the abstraction level l . Let $f(\mathbf{x})$ be a formula of L whose free variables are \mathbf{x} . The supremum $sup^l(f(\mathbf{x}))$ of $f(\mathbf{x})$ at the abstraction level l is a set of tuples a^l in $(D_l)^n$ defined as:

$$sup^l(f(\mathbf{x})) \stackrel{\text{def}}{=} \{a^l : \vdash CDB_l \rightarrow \exists \mathbf{x}(a^l(\mathbf{x}) \wedge f(\mathbf{x}))\}$$

Definition 2. Infimum of f at the abstraction level l . Let $f(\mathbf{x})$ be a formula of L whose free variables are \mathbf{x} . The infimum $inf^l(f(\mathbf{x}))$ of $f(\mathbf{x})$ at the abstraction level l is a set of tuples a^l in $(D_l)^n$ defined as:

$$inf^l(f(\mathbf{x})) \stackrel{\text{def}}{=} \{a^l : \vdash CDB_l \rightarrow \forall \mathbf{x}(a^l(\mathbf{x}) \rightarrow f(\mathbf{x}))\}$$

We shall use the notations:

$$S^l(\mathbf{x}) \stackrel{\text{def}}{=} \bigvee_{a^l \in sup^l(f(\mathbf{x}))} a^l(\mathbf{x})$$

$$I^l(\mathbf{x}) \stackrel{\text{def}}{=} \bigvee_{a^l \in inf^l(f(\mathbf{x}))} a^l(\mathbf{x})$$

Proposition 3. *We have the following properties:*

$$\vdash CDB_l \rightarrow \forall \mathbf{x}(I^l(\mathbf{x}) \rightarrow f(\mathbf{x}))$$

$$\vdash CDB_l \rightarrow \forall \mathbf{x}(f(\mathbf{x}) \rightarrow S^l(\mathbf{x}))$$

The intuitive interpretation of proposition 3 is that tuples in $sup^l(f(\mathbf{x}))$ and in $inf^l(f(\mathbf{x}))$ respectively define an upper bound and a lower bound of $f(\mathbf{x})$.

Definition 4. Upper bound of f at the abstraction level l . Let $f(\mathbf{x})$ be a formula of L whose free variables are \mathbf{x} . An upper bound $upp^l(f(\mathbf{x}))$ at the abstraction level l is any subset of $(D_l)^n$ such that $sup^l(f(\mathbf{x})) \subseteq upp^l(f(\mathbf{x}))$.

Definition 5. Lower bound of f at the abstraction level l . Let $f(\mathbf{x})$ be a formula of L whose free variables are \mathbf{x} . A lower bound $low^l(f(\mathbf{x}))$ at the abstraction level l is any subset of $(D_l)^n$ such that $low^l(f(\mathbf{x})) \subseteq inf^l(f(\mathbf{x}))$.

3 Algebraic computation of abstract answers

Definition 6. Algebra at the abstraction level l . An algebra at the abstraction level l is defined on subsets of $(D_l)^n$. The operators are the operators of the relational algebra: \cup (union), $-$ (difference), \times (cartesian product), Π_i (projection), $P_{I/J}$ (permutation) and S_σ (selection). The only difference with respect to standard relational algebra is that operands of operators of the form $(D_l)^m$ are allowed.

For convenience, here we do not use the standard notation for the projection operator. A tuple of the form $\langle a_1^l, \dots, a_{i-1}^l, a_{i+1}^l, \dots, a_n^l \rangle$ is in $\Pi_i(E)$ iff there exists some a_i^l in D_l such that $\langle a_1^l, \dots, a_{i-1}^l, a_i^l, a_{i+1}^l, \dots, a_n^l \rangle$ is in E . If $I = \langle i_1, \dots, i_p \rangle$, we use the notation $\Pi_I(E)$ to denote $\Pi_{i_1}(\dots \Pi_{i_p}(E) \dots)$.

The permutation operator is defined as follows. Let I and J be two tuples of p index values $I = \langle i_1, \dots, i_p \rangle$ and $J = \langle j_1, \dots, j_p \rangle$, such that there exists a one to one mapping between the I components and the J components. Let $\langle a_1^l, \dots, a_p^l \rangle$ be a tuple in E , the corresponding tuple in $P_{I/J}(E)$ is defined as follows: for s in $[1, p]$, if i_m is such that i_m and j_s denote the same index value, then a_m^l is at the position s in the result of the permutation.

For instance, if we have $I = \langle 3, 5, 2, 6 \rangle$ and $J = \langle 2, 3, 5, 6 \rangle$, we have $j_1 = i_3, j_2 = i_5, j_3 = i_2$ and $j_4 = i_6$. Then, the result of the permutation $P_{I/J}$, when applied to the tuple $\langle a_1^l, a_2^l, a_3^l, a_4^l \rangle$, is $\langle a_3^l, a_1^l, a_2^l, a_4^l \rangle$.

We shall use the following notations in the presentation of propositions 7 and 8. Let $f(\mathbf{x})$ and $g(\mathbf{y})$ be formulas of L whose free variables respectively are $\mathbf{x} = \langle x_{i_1}, \dots, x_{i_p} \rangle$ and $\mathbf{y} = \langle x_{k_1}, \dots, x_{k_r} \rangle$.

Let \mathbf{z} be the tuple of variables $\mathbf{z} = \langle x_{j_1}, \dots, x_{j_q} \rangle$ such that $\{x_{j_1}, \dots, x_{j_q}\} = \{x_{i_1}, \dots, x_{i_p}\} \cup \{x_{k_1}, \dots, x_{k_r}\}$, and $j_1 < j_2 < \dots < j_q$. We use the notations:

$I_1 = \langle i_1, \dots, i_p \rangle$, $K_1 = \langle k_1, \dots, k_r \rangle$, $J = \langle j_1, \dots, j_q \rangle$,

$I_2 = \langle i'_1, \dots, i'_{q-p} \rangle$ such that $\{i'_1, \dots, i'_{q-p}\} = \{j_1, \dots, j_q\} - \{i_1, \dots, i_p\}$ and $i'_1 < i'_2 < \dots < i'_{q-p}$.

$K_2 = \langle k'_1, \dots, k'_{q-r} \rangle$ such that $\{k'_1, \dots, k'_{q-r}\} = \{j_1, \dots, j_q\} - \{k_1, \dots, k_r\}$ and $k'_1 < k'_2 < \dots < k'_{q-r}$.

$I = \langle I_1, I_2 \rangle$ and $K = \langle K_1, K_2 \rangle$.

For instance, if we have: $\mathbf{x} = \langle x_3, x_5 \rangle$ and $\mathbf{y} = \langle x_2, x_3, x_6 \rangle$, we have $\mathbf{z} = \langle x_2, x_3, x_5, x_6 \rangle$, and

$I_1 = \langle i_1, i_2 \rangle = \langle 3, 5 \rangle$, $K_1 = \langle k_1, k_2, k_3 \rangle = \langle 2, 3, 6 \rangle$,

$J = \langle j_1, j_2, j_3, j_4 \rangle = \langle 2, 3, 5, 6 \rangle$,

$I_2 = \langle i'_1, i'_2 \rangle = \langle 2, 6 \rangle$, $K_2 = \langle k'_1 \rangle = \langle 5 \rangle$,

$I = \langle 3, 5, 2, 6 \rangle$ and $K = \langle 2, 3, 5, 6 \rangle$.

Proposition 7. *The supremum and infimum of non atomic formulas at the abstraction level l satisfy the following properties.*

$$(1) \quad \text{sup}^l(f(\mathbf{x}) \vee g(\mathbf{y})) = P_{I/J}(\text{sup}^l(f(\mathbf{x})) \times (D_l)^{q-p}) \cup P_{K/J}(\text{sup}^l(g(\mathbf{y})) \times (D_l)^{q-r})$$

$$(2) \quad P_{I/J}(\text{inf}^l(f(\mathbf{x})) \times (D_l)^{q-p}) \cup P_{K/J}(\text{inf}^l(g(\mathbf{y})) \times (D_l)^{q-r}) \subseteq \text{inf}^l(f(\mathbf{x}) \vee g(\mathbf{y}))$$

$$(3) \quad \text{sup}^l(\neg f(\mathbf{x})) = (D_l)^p - \text{inf}^l(f(\mathbf{x}))$$

$$(4) \quad \text{inf}^l(\neg f(\mathbf{x})) = (D_l)^p - \text{sup}^l(f(\mathbf{x}))$$

$$(5) \quad \text{sup}^l(\exists x_{i_m} f(\mathbf{x})) = \Pi_m(\text{sup}^l(f(\mathbf{x})))$$

$$(6) \quad \Pi_m(\text{inf}^l(f(\mathbf{x}))) \subseteq \text{inf}^l(\exists x_{i_m} f(\mathbf{x}))$$

We also have:

$$(7) \quad \text{sup}^l(f(\mathbf{x}) \wedge g(\mathbf{y})) \subseteq P_{I/J}(\text{sup}^l(f(\mathbf{x})) \times (D_l)^{q-p}) \cap P_{K/J}(\text{sup}^l(g(\mathbf{y})) \times (D_l)^{q-r})$$

$$(8) \quad \text{inf}^l(f(\mathbf{x}) \wedge g(\mathbf{y})) = P_{I/J}(\text{inf}^l(f(\mathbf{x})) \times (D_l)^{q-p}) \cap P_{K/J}(\text{inf}^l(g(\mathbf{y})) \times (D_l)^{q-r})$$

$$(9) \quad \text{sup}^l(\forall x_{i_m} f(\mathbf{x})) \subseteq (D_l)^{p-1} - \Pi_m((D_l)^p - \text{sup}^l(f(\mathbf{x})))$$

$$(10) \quad \text{inf}^l(\forall x_{i_m} f(\mathbf{x})) = (D_l)^{p-1} - \Pi_m((D_l)^p - \text{inf}^l(f(\mathbf{x})))$$

Notice that if f and g have the same free variables, that is $\mathbf{x} = \mathbf{y}$, we have $I = K = J$, and $p = r = q$. Since we have $P_{I/I}(E) = E$ and $E \times (D_l)^0 = E$, we have the simpler properties:

$$(1) \quad \text{sup}^l(f(\mathbf{x}) \vee g(\mathbf{x})) = \text{sup}^l(f(\mathbf{x})) \cup \text{sup}^l(g(\mathbf{x}))$$

$$(2) \quad \text{inf}^l(f(\mathbf{x})) \cup \text{inf}^l(g(\mathbf{x})) \subseteq \text{inf}^l(f(\mathbf{x}) \vee g(\mathbf{x}))$$

Properties (7) and (8) simplify in the same way.

The reason why in (2) and (6) we have an inclusion instead of an equality can be justified with simple counter examples.

Let's consider, for example, the formulas $f(x_1) = \text{weather}(\text{Strasbourg}, x_1, \text{raining})$ and $g(x_1) = \text{weather}(\text{Strasbourg}, x_1, \text{snowing})$, and $f(x_1) \vee g(x_1)$, which defines on which days it is either raining or snowing in Strasbourg. If January belongs to $\text{inf}^l(f(x_1) \vee g(x_1))$, that is, if on every days in January it is either raining or snowing in Strasbourg, it is not necessarily the case that either on every days in January it is raining in Strasbourg, or on every days on January it is snowing in Strasbourg. That means that January does not necessarily belong to $\text{inf}^l(f(x_1))$ nor to $\text{inf}^l(g(x_1))$. That is why we do not have equality in (2).

Let's consider now the formula $f'(x_1, x_2) = \text{weather}(x_1, x_2, \text{foggy})$, which defines in which cities and on which days it is foggy. If the county Gironde belongs to $\text{inf}^l(\exists x_2 \text{weather}(x_1, x_2, \text{foggy}))$, that is, if for every cities in Gironde there exists a day on which it is foggy, it is not necessarily the case that there exists a month such that for every cities in Gironde and for every days on this month it is foggy in this city on this day. That means that it does not necessarily exist a month M such that $\langle \text{Gironde}, M \rangle$ belongs to $\text{inf}^l(\text{weather}(x_1, x_2, \text{foggy}))$, and Gironde does not necessarily belong to $\Pi_1(\text{inf}^l(\text{weather}(x_1, x_2, \text{foggy})))$. That is why we do not have equality in (6).

One can notice the formal analogy between the definitions of sup^l (resp. inf^l) and of the necessity operator (resp. the possibility operator) in modal logic. In (2) and (7) we do not have equality for the same reason as in modal logic the necessity operator cannot be distributed on the disjunction, and the possibility operator cannot be distributed on the conjunction operator.

Proposition 8. *Let $h(\mathbf{x})$ be a non atomic formula of L . An upper bound $\text{upp}^l(h(\mathbf{x}))$ of $h(\mathbf{x})$ and a lower bound $\text{low}^l(h(\mathbf{x}))$ of $h(\mathbf{x})$ can be computed with the following formulas.*

$$\text{upp}^l(f(\mathbf{x}) \vee g(\mathbf{y})) = P_{I/J}(\text{upp}^l(f(\mathbf{x})) \times (D_l)^{q-p}) \cup P_{K/J}(\text{upp}^l(g(\mathbf{y})) \times (D_l)^{q-r})$$

$$\text{low}^l(f(\mathbf{x}) \vee g(\mathbf{y})) = P_{I/J}(\text{low}^l(f(\mathbf{x})) \times (D_l)^{q-p}) \cup P_{K/J}(\text{low}^l(g(\mathbf{y})) \times (D_l)^{q-r})$$

$$\text{upp}^l(\neg f(\mathbf{x})) = (D_l)^p - \text{low}^l(f(\mathbf{x}))$$

$$\text{low}^l(\neg f(\mathbf{x})) = (D_l)^p - \text{upp}^l(f(\mathbf{x}))$$

$$\begin{aligned} \text{upp}^l(\exists x_{i_m} f(\mathbf{x})) &= \Pi_m(\text{upp}^l(f(\mathbf{x}))) \\ \text{low}^l(\exists x_{i_m} f(\mathbf{x})) &= \Pi_m(\text{low}^l(f(\mathbf{x}))). \end{aligned}$$

We also have:

$$\begin{aligned} \text{upp}^l(f(\mathbf{x}) \wedge g(\mathbf{y})) &= P_{I/J}(\text{upp}^l(f(\mathbf{x})) \times (D_i)^{q-p}) \cap P_{K/J}(\text{upp}^l(g(\mathbf{y})) \times (D_i)^{q-r}) \\ \text{low}^l(f(\mathbf{x}) \wedge g(\mathbf{y})) &= P_{I/J}(\text{low}^l(f(\mathbf{x})) \times (D_i)^{q-p}) \cap P_{K/J}(\text{low}^l(g(\mathbf{y})) \times (D_i)^{q-r}) \\ \text{upp}^l(\forall x_{i_m} f(\mathbf{x})) &= (D_i)^{p-1} - \Pi_m((D_i)^p - \text{upp}^l(f(\mathbf{x}))) \\ \text{low}^l(\forall x_{i_m} f(\mathbf{x})) &= (D_i)^{p-1} - \Pi_m((D_i)^p - \text{low}^l(f(\mathbf{x}))). \end{aligned}$$

Definition9. Extension of a formula f . Let $f(\mathbf{x})$ be a formula of L whose free variables are \mathbf{x} . The extension $\text{ext}(f(\mathbf{x}))$ of f is defined as:

$$\text{ext}(f(\mathbf{x})) \stackrel{\text{def}}{=} \{c : c \in D^p \text{ and } \vdash CDB \rightarrow f(c)\}$$

The cardinality of $\text{ext}(f(\mathbf{x}))$ is denoted by $|f(\mathbf{x})|$.

Definition10. Standard answer to a query q . Let $q(\mathbf{x})$ be a formula of L whose free variables are \mathbf{x} . The standard answer $\text{ans}(q(\mathbf{x}))$ to the query q is its extension. We have: $\text{ans}(q(\mathbf{x})) \stackrel{\text{def}}{=} \text{ext}(q(\mathbf{x}))$.

Definition11. Abstract answer to a query q at the level l . Let $q(\mathbf{x})$ be a formula of L whose free variables are \mathbf{x} . The abstract answer $\text{ans}^l(q(\mathbf{x}))$ to the query q at the level l is defined as:

$$\text{ans}^l(q(\mathbf{x})) \stackrel{\text{def}}{=} \{a^l : a^l \in \text{upp}^l(q(\mathbf{x})) - \text{low}^l(q(\mathbf{x})) \text{ and } |a^l(\mathbf{x}) \wedge q(\mathbf{x})| \geq |a^l(\mathbf{x}) \wedge \neg q(\mathbf{x})|\} \cup \text{low}^l(q(\mathbf{x}))$$

Notice that every a^l which is in $\text{upp}^l(q(\mathbf{x})) - \text{low}^l(q(\mathbf{x}))$, and such that the extensions of $a^l(\mathbf{x})$ and $q(\mathbf{x})$ have no common element, are not in $\text{ans}^l(q(\mathbf{x}))$, because $|a^l(\mathbf{x}) \wedge q(\mathbf{x})| = 0$.

Definition12. Refinement of a set of tuples of abstract elements. Let E be a set of tuples of abstract elements, that is $E \subseteq (D_i)^p$. The refinement $\text{ref}(E)$ of E is defined as:

$$\text{ref}(E) \stackrel{\text{def}}{=} \bigvee_{a^l \in E} a^l(\mathbf{x})$$

Definition13. Error of a set of abstract tuples with respect to the standard answer to a query. Let E be a set of abstract tuples, and let $q(\mathbf{x})$ be a query. The error $\text{err}(E, q(\mathbf{x}))$ of E with respect to the standard answer to $q(\mathbf{x})$ is defined as the set of tuples in D^p that are in the extension of $q(\mathbf{x})$ and that are not in the refinement of E , plus the tuples that are in the refinement of E and that are not in the extension of $q(\mathbf{x})$. That is, we have:

$$\text{err}(E, q(\mathbf{x})) \stackrel{\text{def}}{=} \text{ext}(q(\mathbf{x})) \wedge \neg \text{ref}(E) \vee \text{ref}(E) \wedge \neg q(\mathbf{x})$$

Proposition14. *The cardinality of the error of the abstract answer at the level l to a query $q(\mathbf{x})$ with regard to the extension of $q(\mathbf{x})$ is lower than the error of the upper bound of $q(\mathbf{x})$ (computed by the formulas given in proposition 8) with regard to the extension of $q(\mathbf{x})$. That is, we have:*

$$|\text{err}(\text{ans}^l(q(\mathbf{x})), q(\mathbf{x}))| \leq |\text{err}(\text{upp}^l(q(\mathbf{x})), q(\mathbf{x}))|$$

4 Implementation guidelines

The implementation of the notions we have presented in the previous section depends on the “quality” of the answer we want to communicate to users.

The first option is to only communicate the upper bound and the lower bound that can be computed using proposition 8. In that case users know that all the elements in the refinement of the lower bound are in the standard answer, but some of them may be missing. At the opposite, all the elements in the standard answer are in the refinement of the upper bound, but some of the elements in the upper bound may not be in the standard answer. That is, the lower bound characterises valid answers, but not necessarily complete answers, while the upper bound characterises complete answers that are not necessarily valid answers.

The second option is to communicate to users the abstract answer (see Definition 12). This answer is better than the upper bound and the lower bound, in the sense that the error is reduced. However, users have no guarantee about its validity nor about its completeness.

The third option is to communicate, in addition to the abstract answer, the tuples that have to be added, that is the extension of $q(\mathbf{x}) \wedge \neg ans^l(q(\mathbf{x}))$, and the tuples that have to be removed, that is the extension of $ans^l(q(\mathbf{x})) \wedge \neg q(\mathbf{x})$. Then, users know a characterisation of the correct answer. Part of it is at the abstract level l , and part of it is at the concrete level. Of course, if the abstraction level increases, the size of the abstract part decreases, but the size of the concrete part increases. Optimal tradeoff have to be found for each application domain.

In the first option we have to compute an upper bound and a lower bound at the abstract level l of a given query $q(\mathbf{x})$, provided we know an upper bound and a lower bound of atomic sentences of the form $p(t_1, \dots, t_n)$ that occur in q , where the t_i s may be constant symbols or variable symbols.

One possibility is to directly compute for these atomic sentences the supremum or the infimum, and to take these values as upper bound or lower bound, that is: $upp^l(p(t_1, \dots, t_n)) = sup^l(p(t_1, \dots, t_n))$ and $low^l(p(t_1, \dots, t_n)) = inf^l(p(t_1, \dots, t_n))$. However, this computation may be expensive and, since it has to be performed at the moment the query has been asked, it may lead to long response time.

Another possibility is to compute, before the query has been asked the supremum and the infimum of atomic sentences of the form $p(x_1, \dots, x_n)$, without “constraints”, in the sense that no t_i is a constant symbol and there is no variable symbol that occurs several times. Notice that there is only one atomic sentence of this form for each predicate symbol. Let’s respectively call A_i^+ and A_i^- the supremum and the infimum, that is: $A_i^+ = sup^l(p(x_1, \dots, x_n))$ and $A_i^- = inf^l(p(x_1, \dots, x_n))$.

Then, for atomic sentences of the form $p(t_1, \dots, t_n)$ with constraints we can compute an upper bound and a lower bound with the algebraic formulas:

$$upp^l(p(t_1, \dots, t_n)) = P_{I/J}(\Pi_{I'}(S_\sigma(A_i^+)))$$

$$low^l(p(t_1, \dots, t_n)) = P_{I/J}(\Pi_{I'}(S_\sigma(A_i^-)))$$

where σ , I' , I and J are defined as follows.

The selection condition σ is a conjunction of atomic conditions. If t_s is the constant symbol c_{i_s} , then the condition $(s = c_{i_s})$ is in σ . If t_s is the variable symbol x_{i_s} , and there exists s' such that $t_s = t_{s'}$, then $s = s'$ is in σ . There is no other condition in σ .

Let I and I' be the index tuples $I = \langle i_{s_1}, \dots, i_{s_p} \rangle$ and $I' = \langle s_1, \dots, s_p \rangle$ such that for r in $[1, p]$ we have: t_{s_r} is a variable symbol $x_{i_{s_r}}$ and there is no s' such that $s' < s$ and $t_{s_r} = t_{s'}$. Then, we have $J = \langle j_1, \dots, j_p \rangle$ such that there is a one to one mapping between I and J , and we have $j_1 < j_2 < \dots < j_p$.

For instance, if we have $p(t_1, \dots, t_n) = p(x_3, c, x_2, x_3)$, we have $t_1 = t_4 = x_3$ and $t_2 = c$, then we have $\sigma = (1 = 4) \wedge (2 = c)$. We also have: $s_1 = 1$, $t_{s_1} = x_{i_{s_1}} = x_3$ and $i_{s_1} = 3$, and $s_2 = 3$, $t_{s_2} = x_{i_{s_2}} = x_2$ and $i_{s_2} = 2$. Then, we have: $I' = \langle s_1, s_2 \rangle = \langle 1, 3 \rangle$, $I = \langle i_{s_1}, i_{s_2} \rangle = \langle 3, 2 \rangle$ and $J = \langle 2, 3 \rangle$. We finally have: $upp^l(p(x_3, c, x_2, x_3)) = P_{\langle 3, 2 \rangle / \langle 2, 3 \rangle}(\Pi_{\langle 1, 3 \rangle}(S_{(1=4) \wedge (2=c)}(A_l^+)))$ and $low^l(p(x_3, c, x_2, x_3)) = P_{\langle 3, 2 \rangle / \langle 2, 3 \rangle}(\Pi_{\langle 1, 3 \rangle}(S_{(1=4) \wedge (2=c)}(A_l^-)))$.

In the second option, in addition to $upp^l(q(\mathbf{x}))$ and $low^l(q(\mathbf{x}))$, for each tuple a^l in $upp^l(q(\mathbf{x})) - low^l(q(\mathbf{x}))$ we have to compute the extension of $a^l(\mathbf{x}) \wedge q(\mathbf{x})$ and of $a^l(\mathbf{x}) \wedge \neg q(\mathbf{x})$. Therefore, we need to compute the extension of $q(\mathbf{x})$, that is the standard answer.

In the third option, in addition to $ans^l(q(\mathbf{x}))$, we have to compute the extension of $q(\mathbf{x}) \wedge \neg ref(ans^l(q(\mathbf{x})))$ and of $ref(ans^l(q(\mathbf{x}))) \wedge \neg q(\mathbf{x})$. Since the extension of $q(\mathbf{x})$ has been already computed in the computation of the abstract answer, the additional cost is basically due to the computation of the extension of $ref(ans^l(q(\mathbf{x})))$.

A practical problem raised by the implementation of the extended relational algebra presented in section 3 is that algebraic expressions may contain cartesian products whose operands are relation domains. That can lead to extremely expensive computation costs. Then, we need to restrict queries to those ones whose corresponding algebraic computation expressions can be translated into the standard relational algebra in order to remove these cartesian products.

In fact the presented extended relational algebra is a variant of the cylindric algebra [IL81]. It has been shown by J.F. Ullman in [Ull80] that, if we restrict queries to queries that are "domain independent" [Dem82] there exists a translation into the relational algebra. However, since the class of domain independent formulas is not decidable [Pao69], we have to restrict them to a decidable class. In [Dem82] we have defined such decidable class whose characterisation is based on constraint on variables that occur in queries (corresponding queries are called "evaluable" queries). We have no room here to present their definition but we can have an intuitive idea from the fact that "range restricted" queries [Nic82] are just a special case of evaluable queries.

To intuitively understand why for evaluable queries we can obtain a translation into standard relational algebra we present the following example.

Let's consider the evaluable query: $q(x_1, x_2) \wedge \neg r(x_2)$. From proposition 8 we have: $upp^l(q(x_1, x_2) \wedge \neg r(x_2)) = P_{\langle 1, 2 \rangle / \langle 1, 2 \rangle}(upp^l(q(x_1, x_2)) \times D_l^0) \cap P_{\langle 2, 1 \rangle / \langle 1, 2 \rangle}(upp^l(\neg r(x_2)) \times D_l)$. Let's assume that: $upp^l(q(x_1, x_2)) = Q^+$ and $low^l(r(x_2)) = R^-$.

We have $P_{\langle 1,2 \rangle / \langle 1,2 \rangle}(\text{upp}^l(q(x_1, x_2)) \times D_l^0) = P_{\langle 1,2 \rangle / \langle 1,2 \rangle}(Q^+) = Q^+$. We also have: $\text{upp}^l(\neg r(x_2)) = D_l - \text{low}^l(r(x_2)) = D_l - R^-$. Then $P_{\langle 2,1 \rangle / \langle 1,2 \rangle}(\text{upp}^l(\neg r(x_2)) \times D_l) = P_{\langle 2,1 \rangle / \langle 1,2 \rangle}((D_l - R^-) \times D_l) = D_l \times (D_l - R^-) = D_l^2 - (D_l \times R^-)$.

Finally, we have: $\text{upp}^l(q(x_1, x_2) \wedge \neg r(x_2)) = Q^+ \cap (D_l^2 - (D_l \times R^-)) = Q^+ - (D_l \times R^-) = Q^+ - (\Pi_1(Q^+) \times R^-)$. We can see that the final expression does not contain cartesian products with relations domains. We could use optimisation techniques to transform it into $Q^+ - \Pi_{\langle 1,2 \rangle}(Q^+ \otimes_{1=1} R^-)$ that contains no cartesian product.

5 Conclusion

We have presented a formal definition of abstract answers, of their lower bounds and of their upper bounds. We have also presented a method to algebraically compute lower bounds and upper bounds. In the case where a user wants to know an exact answer we have defined a method to reduce the error between the refinement of the abstract answer and the standard answer. Indeed, the user can get an answer made of the elements in the error (i.e. in $\text{err}(\text{upp}^l(q(\mathbf{x})), q(\mathbf{x}))$) that have to be removed or to be added to the refinement of the abstract answer.

The proofs of the propositions are not given in the paper due to the lack of room, but they are not very hard, we just need to carefully use the definitions.

The results are quite general in the sense that a database is not restricted to be a standard Relational Database, it can be as well a Deductive Database.

References

- [Dem82] R. Demolombe. Syntactical Characterization of a Subset of Domain Independent Formulas. *Journal of ACM*, 1982.
- [Ell95] T. Ellman. Approximation and Abstraction Techniques for Generating Concise Answers to Database Queries. In *Proc. of AAAI Spring Symposium*, 1995.
- [IL81] T. Imielinski and W. Lipski. The Relational model of data and cylindric algebras. Technical Report 446, Institute of Computer Science, Polish Academy of Science, 1981.
- [Imi87] T. Imielinski. Domain abstraction and limited reasoning. In *Proc. of the 10th IJCAI*, 1987.
- [Nic82] J-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, Vol 18(Num 3), 1982.
- [Pao69] R. A. Di Paola. The recursive unsolvability of the decision problem for the class of Definite Formulas. *Journal of ACM*, 16(2), 1969.
- [Ull80] J. D. Ullman. *Principles of Database Systems*. Computer Science Press, 1980.