Cooperative Query Answering by Abstract Interpretation

Raju Halder and Agostino Cortesi

Dipartimento di Informatica
Università Ca' Foscari Venezia, Italy
{halder,cortesi}@unive.it

Abstract. A common problem for many database users is how to formulate and submit correct queries in order to get useful responses from the system, with little or no knowledge of the database structure and its content. The notion of cooperative query answering has been explored as an effective mechanism to address this problem. In this paper, we propose a cooperative query answering scheme based on the Abstract Interpretation framework. In this context, we address three key issues: soundness, relevancy and optimality of the cooperative answers.

Keywords: Databases, Cooperative Query Answering, Abstract Interpretation.

1 Introduction

Traditional query processing system enforces database-users to issue precisely specified queries, while the system provides limited and exact answers, or no information at all when the exact answer is unavailable in the database. Therefore, it is important to the database-users to fully understand problem domain, query syntax, database schema, and underlying database content.

To remedy such shortcomings and to enhance the effectiveness of the information retrieval, the notion of cooperative query answering [5,6,8,15] has been explored. The cooperative query answering system provides users an intelligent database interface that allows them to issue approximate queries independent to the underlying database structure and its content, and provides additional useful information as well as the exact answers.

As an example, in response to the query about "specific flight departing at *10 a.m.* from *Rome Fiumicino* airport to *Paris Orly* airport" the cooperative query answering system may return "all flight information during *morning* time from airports in *Rome* to airports in *Paris*", and thus, the user will be able to choose other flight if the specific flight is unavailable. Such query answering is also known as neighborhood query answering, as instead of providing exact answers it captures neighboring information as well. Cooperative query answering system also gives users the opportunity to issue conceptual or approximate queries where they might ask more general questions, for example, "how to travel from *Rome* to *Paris* at a *reasonable* cost during *morning* time" or "find the flights that fly

I. Černá et al. (Eds.): SOFSEM 2011, LNCS 6543, pp. 284–296, 2011.

during *night* only" without knowing the exact database schema and its content. One of the benefits of issuing conceptual queries is to avoid reissuing of the set of concrete queries if the corresponding conceptual query returns empty result.

Cooperative query answering depends on the context in which queries are issued. The context includes the identity of the issuer, the intent or purpose of the query, the requirements that make explicit the answers relevant to the user etc. The following example illustrates it clearly: suppose a user issues a query asking the list of airports that are similar to "*Venice Marcopolo*" airport. Different contexts define the meaning of "*similarity between airports*" differently. For instance, to any surveyor "*similarity*" may refer in terms of the size and facilities provided in the airport, whereas to any flight company "*similarity*" may refer in terms of business point of view *i.e.* flight landing charges or other relevant taxes.

Abstract Interpretation is a well known semantics-based static analysis technique [7,9,11], originally developed by Cousot and Cousot as a unifying framework for designing and then validating static program analysis, and recently it becomes a general methodology for describing and formalizing approximate computation in many different areas of computer science, like model checking, process calculi, security, type inference, constraint solving, etc [9].

In our previous work [11], we introduced Abstract Interpretation framework to the field of database query languages as a way to provide sound approximation of the query languages. In this paper, we extend this to the field of cooperative query answering system: we propose a cooperative query answering scheme based on the Abstract Interpretation framework that consists of three phases - transformation of the whole query system by using abstract representation of the data values, cooperative query evaluation over the abstract domain, and concretization of the cooperative abstract result. The main contributions in this paper are: (*i*) we express the cooperative query evaluation by abstract databases, (*ii*) we express how to deal with cooperative query evaluation in presence of aggregate and negation operations, (*iii*) we address three key issues: soundness, relevancy and optimality of the cooperative answers.

The structure of the paper is as follows: Section 2 discusses related work in the literature and motivation of our work. Section 3 describes the key issues in the context of cooperative query answering. In Section 4, we discuss our proposed scheme. In Section 5, we show how our proposal is able to address the key issues. Finally, we draw our conclusions in Section 6.

2 Related Work and Motivation

Several techniques have been proposed in the literature based on logic model, semantic distance, fuzzy set theory, abstraction, and so on. The logic-based models [1,8] use first-order predicate logic to represent the database, the knowledge-base, and the user's queries. Content of the knowledge base helps in guiding query reformulation into more flexible and generalized query that provides relaxed, intelligent cooperative answer. However, these approaches have limitations in guiding

the query relaxation process and the less intuitive query answering process due to lack of its expressiveness.

The semantic distance-based approaches [14,17] use the notion of semantic distance to represent the degree of similarity between data values, and provide ranked cooperative results sorted according to their semantic distances. For categorical data, distances between data values are stored in a table. However, since every pair is supposed to have semantic distances, in realistic application domain these approaches are inefficient as the table size gets extremely larger and becomes harder to maintain the consistency of the distance measures.

In [10], the initial queries are transformed into flexible form based on knowledge base and fuzzy set theory. Finally, these queries are rewritten into boolean queries and evaluated to the traditional database.

In abstraction-based models [4,6], the data are organized into multilevel abstraction hierarchies where nodes at higher level are the abstract representation of the nodes at lower level. The cooperative query answering is accomplished by generating a set of refined concrete queries by performing query abstraction and query refinement process by moving upward or downward through the hierarchy. Finally, the refined queries are issued to the database that provide additional useful information. These approaches suffer from high overhead when the degree of relaxation for a query is large, as the query abstraction-refinement process produces a large set of concrete queries to be issued to the database. To remedy this, fuzzy set theory or semantic distance approach is combined with abstraction hierarchy [5,13,15] to control the abstraction-refinement process and to provide a measure of nearness between exact and approximate answers.

However, all the above mentioned schemes do not provide any formal framework to cooperative query answering system. In addition, none of these schemes enlightens the key issues: soundness, relevancy and optimality in the context of cooperativeness of the query answers. Most of the existing techniques [4,5,6,13,15] suffer from the problem of soundness when query contains negation operation MINUS. In case of conceptual or approximate queries where approximate results are desirable, none of the schemes focuses on the way to compute aggregate functions when appearing in a query so as to preserve the soundness.

3 Key Issues: Soundness, Relevancy, Optimality

Any cooperative query answering scheme should respect three key issues: soundness, relevancy, and optimality. Intuitively, a cooperative query answer is *sound* if it is equal to or it is a superset of the corresponding extensional query answer. The *relevancy* of the answers *w.r.t.* the context concerns with avoiding the tuples that have no value to the user: all the information in the cooperative answer should have relevancy to the user. The third criteria *optimality* implies that the system should return as much information as possible, while satisfying the first two properties. Since there exist many cooperative answers corresponding to a given query under given context, the optimality describes the "preferability" of the query answers to the user.

Given a cooperative query processing system $\mathfrak{B}$, a database dB, a context C, and a query Q, the result obtained by the cooperative system is $R = \mathfrak{B}(dB, C, Q)$.

Definition 1 (Soundness). *Given a database dB, a query Q, and a context C. Let R be the extensional query answer obtained by processing Q on dB. The cooperative answer $R' = \mathfrak{B}(dB, C, Q)$ is sound if $R' \supseteq R$.*

The relevancy of the information to a user depends on his interests. These interests can be expressed in terms of constraints represented by well-formed formulas in first order logic. If the information provided by a cooperative system satisfies the set of constraints representing user's interests, it is treated as relevant. For instance, "flight duration in the result set must be less than 3 hours" can be used as a constraint that determines the relevancy of the information in the cooperative answer.

Definition 2 (Relevancy). *Given a database dB, a query Q, and a context C. Let $S(Q)$ be the set of constraints represented by well-formed formulas in first order logic that make explicit the answers relevant to the user. The cooperative answer $R = \mathfrak{B}(dB, C, Q)$ respects the relevancy if $\forall x \in R : \ x \models S(Q)$.*

It is worthwhile to mention that the system relaxes users' queries to obtain neighboring and conceptually related answers in addition to the exact answer. However, the formulas appearing in the pre-condition ϕ [11] of the query Q is different from the set of constraints appearing in $S(Q)$ that determines the relevancy of the cooperative answers. The constraints in $S(Q)$, in contrast to ϕ, is strict in the sense that there is no question of relaxing them, and violation of any of these constraints by the information in the cooperative answer will be treated as irrelevant.

A cooperative system may return different cooperative answers to a user in a given context. However, it is sensible to define a measure that describes the "preferability" of each answer. A cooperative answer is called more optimal than another answer if it is more preferable to the user in the given context than the other.

Definition 3 (Optimality). *Given a database dB, a query Q, a context C, and a set of constraints $S(Q)$ expressing user's requirements. The cooperative answer $R = \mathfrak{B}(dB, C, Q)$ is more optimal than $R' = \mathfrak{B}'(dB, C, Q)$ if*

$$\{x \in R : \ x \models S(Q)\} \supseteq \{x \in R' : \ x \models S(Q)\} \text{ and } \{y \in R : \ y \not\models S(Q)\} \subseteq \{y \in R' : \ y \not\models S(Q)\}$$

In other words, a cooperative answer is called more optimal than another answer when it contains more relevant information and less irrelevant information *w.r.t.* $S(Q)$ than the other.

4 Proposed Scheme

Our proposal consists of three phases: (*i*) Transforming the databases and its query languages by using abstract representation of the data values, (*ii*) Cooperative query evaluation over the abstract domain, and finally, (*iii*) Concretization of the cooperative abstract result.

4.1 Transforming from Concrete to Abstract Domain

In this section, we discuss how to lift databases and its query languages from concrete to the abstract domain of interests. The level of approximation of the database information obtained by abstraction gives a measure of the "preferability" of the cooperative answers and depends on the context in which queries are issued. The best correct approximation [9] of the database information according to the context provides the optimal cooperative answers to the end-users.

Generally, traditional databases are *concrete databases* as they contain data from the concrete domain, whereas *abstract databases* are obtained by replacing concrete values by the elements from abstract domains representing specific properties of interests. We may distinguish partially abstract database in contrast to fully abstract one, as in the former case only a subset of the data in the database is abstracted. The values of the data cells belonging to an attribute x are abstracted by following the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$ [9], where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, whereas α_x and γ_x represent the corresponding abstraction and concretization functions (denoted $\alpha_x : \wp(D_x^{con}) \rightarrow D_x^{abs}$ and $\gamma_x : D_x^{abs} \rightarrow \wp(D_x^{con})$) respectively. In particular, partially abstract databases are special case of fully abstract databases where for some attributes x the abstraction and concretization functions are identity functions id, and thus, follow the Galois Connection $(\wp(D_x^{con}), id, id, D_x^{abs})$.

Table 1. Concrete and corresponding Abstract Databases

(a) Database containing concrete table "$flight$"

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F002	Marcopolo (VCE)	Orly (ORY)	410.30	18.30	21.00	Y
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F004	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
F005	Treviso (TSF)	Granby-Grand County (GNB)	200.15	16.00	17.20	Y
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

(b) Abstract database containing abstract table "$flight^\sharp$"

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Venice	Paris	[400.00-449.99]	evening	night	$\top$
$\top$	Rome	Paris	[300.00-349.99]	morning	morning	$\top$
$\top$	Rome	Lyon	[100.00-149.99]	night	night	$\top$
$\top$	Venice	Lyon	[200.00-249.99]	afternoon	evening	$\top$

Let us illustrate it by an example. The database in Table 1(a) consists of concrete table "$flight$" that provides available flight information to the end-users of a travel agent application. The corresponding abstract table "$flight^\sharp$" is shown in Table 1(b) where source and destination airports are abstracted

by the provinces they belong, the numerical values of the cost attribute are abstracted by the elements from the domain of intervals, the values of the start-time/reach-time attributes are abstracted by the periods from the abstract domain $PERIOD = \{\bot, morning, afternoon, evening, night, \top\}$ where $\top$ represents "anytime" and $\bot$ represents "don't know", the flight no. and availability attributes are abstracted by the topmost element $\top$ of their corresponding abstract lattices. Observe that the number of abstract tuples in an abstract database may be less than that in the corresponding concrete database.

Definition 4 (Abstract Database). *Let dB be a database. The database $dB^\sharp = \alpha(dB)$ where α is the abstraction function, is said to be an abstract version of dB if there exist a representation function γ, called concretization function such that for all tuple $\langle x_1, x_2, \ldots, x_n\rangle \in dB$ there exist a tuple $\langle y_1, y_2, \ldots, y_n\rangle \in dB^\sharp$ such that $\forall i \in [1 \ldots n]$ $(x_i \in id(y_i) \vee x_i \in \gamma(y_i))$.*

4.2 Cooperative Query Evaluation

In [11], we proposed denotational semantics of database query languages in both concrete and abstract level. We extend this to the context of cooperative query answering, where the SQL queries are lifted into an abstract setting and instead of working on concrete databases they are applied on the corresponding abstract databases. However, in this paper, we restrict our discussions to the SELECT statements only. Let us start with a simple example:

Example 1. Consider an online booking application interacting with the concrete database depicted in Table 1(a). Suppose a user wants to travel from *Rome Fiumicino* airport to *Paris Orly* airport by a flight such that flight cost is less than or equal to *300 USD*. So the following query satisfying the required criteria can be issued:

```
Q1 =SELECT * FROM flight WHERE source = "Fiumicino" AND destination = "Orly" AND
    cost ≤ 300.00 AND availability=Y;
```

Observe that the result ξ_1 of the query Q_1 is empty *i.e.* $\xi_1 = \emptyset$, because seats are not available in the flight from *Rome Fiumicino* to *Paris Orly* airport.

To obtain cooperative answers, we lift the whole query system from concrete to the abstract domain of interests by abstracting the database information and the associated query languages. The abstract database corresponding to the concrete database (Table 1(a)) is depicted in Table 1(b), and the abstract query corresponding to the concrete query Q_1 is shown below:

```
Q1♯ =SELECT♯ * FROM flight♯ WHERE source♯ =♯ "Rome" AND destination♯ =♯ "Paris" AND
    cost♯ ≤♯ [300.00, 349.99] AND availability♯ =♯ ⊤;
```

where the abstract operation $\leq^\sharp$ for intervals is defined as follows:

$$[l_i, h_i] \leq^\sharp [l_j, h_j] \triangleq \begin{cases} true & \text{if } h_i \leq l_j \\ false & \text{if } l_i > h_j \\ \top & otherwise \end{cases}$$

and the abstract equality $=^\sharp$ is defined as usual.

When the imprecise abstract query $Q_1^\sharp$ is executed over the abstract database (Table 1(b)), it returns the flight information depicted in Table 2. This way, the abstraction of the databases and its query languages helps in obtaining additional information in the result.

Table 2. $\xi_1^\sharp$: Abstract Result of $Q_1^\sharp$

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Rome	Paris	[300.00-349.99]	morning	morning	$\top$

Consider a SELECT statement $Q = \langle A_{sel}, \phi \rangle$ where A_{sel} is *action part* and ϕ is *pre-condition part* of Q [11,12]. In the concrete domain, the pre-condition ϕ evaluates to two-valued logic (*true* and $false$) and the active data set on which A_{sel} operates contains those tuples for which ϕ evaluates to *true* only. In contrast, when we lift the whole query system to an abstract setting, the evaluation of the abstract pre-condition $\phi^\sharp$ over the abstract database results into a three-valued logic (*true*, $false$, and $\top$). The logic value $\top$ indicates that the tuple may or may not satisfy the semantic structure of $\phi^\sharp$. Thus, in the abstract domain, the active data set on which abstract SELECT action $A_{sel}^\sharp$ operates consists of those abstract tuples for which $\phi^\sharp$ evaluates to *true* or $\top$. For instance, in the query result $\xi_1^\sharp$ of Table 2, $\phi^\sharp$ evaluates to *true* for the first tuple with $cost^\sharp$ equal to $[200.00, 249.99]$, whereas it evaluates to $\top$ for the last tuple with $cost^\sharp$ equal to $[300.00, 349.99]$.

Soundness is preserved as the concretization of the abstract queries always results into a sound approximation of the corresponding concrete queries.

Cooperative query evaluation with aggregate functions: In case of conceptual or approximate queries where approximate results are desirable, none of the existing techniques focuses on the way to compute aggregate functions when appearing in a query so as to preserve the soundness. In this section, we discuss how to compute the cooperative aggregate functions in an abstract setting to provide the users a sound approximation of the aggregate results.

Let $T_\phi^\sharp$ be the set of abstract tuples for which $\phi^\sharp$ evaluates to *true* or $\top$. The application of abstract GROUP BY function $g^\sharp$ on $T_\phi^\sharp$ yields to a set of abstract groups $\{G_1^\sharp, G_2^\sharp, \ldots, G_n^\sharp\}$. When no $g^\sharp$ appears in abstract SELECT statement $Q^\sharp$, we assume $T_\phi^\sharp$ as a single abstract group.

Given an abstract group $G^\sharp$, we can partition the tuples in $G^\sharp$ into two parts: $G_{yes}^\sharp$ for which $\phi^\sharp$ evaluates to true, and $G_{may}^\sharp$ for which $\phi^\sharp$ evaluates to $\top$. Thus we can write $G^\sharp = G_{yes}^\sharp \cup G_{may}^\sharp$, where $G_{yes}^\sharp \cap G_{may}^\sharp = \emptyset$.

Let $s^\sharp$ be an abstract aggregate function and $e^\sharp$ be an abstract expression. To ensure the soundness, the computation of $s^\sharp(e^\sharp)$ on $G^\sharp$ is defined as follows: $s^\sharp(e^\sharp)[G^\sharp] = [min^\sharp(a^\sharp),\ max^\sharp(b^\sharp)]$, where $a^\sharp = fn^\sharp(e^\sharp)[G_{yes}^\sharp]$ and $b^\sharp = fn^\sharp(e^\sharp)[G^\sharp]$.

By $fn^\sharp(e^\sharp)[G_{yes}^\sharp]$ and $fn^\sharp(e^\sharp)[G^\sharp]$ we mean that the function $fn^\sharp$ is applied on the set of abstract values obtained by evaluating $e^\sharp$ over the tuples in $G_{yes}^\sharp$ and

$G^\sharp$ respectively, yielding to an abstract aggregate value as result. The computation of $fn^\sharp$ is defined differently by considering two different situations that can arise: (i) when the primary key is abstracted, yielding two or more tuples mapped into a single abstract tuple, and (ii) when the primary key is not abstracted and the identity of each tuples are preserved in abstract domain. Both the functions $min^\sharp$ and $max^\sharp$ take single abstract value $a^\sharp$ and $b^\sharp$ respectively as parameters which are obtained from $fn^\sharp$, and returns a concrete numerical value as output. $min^\sharp(a^\sharp)$ returns minimum concrete value from $\gamma(a^\sharp)$ and $max^\sharp(b^\sharp)$ returns maximum concrete value from $\gamma(b^\sharp)$, where γ is the concretization function.

Lemma 1. *Let s and e be an aggregate function and an expression respectively. Suppose the corresponding abstract representation of s and e are $s^\sharp$ and $e^\sharp$ respectively. Let $s^\sharp(e^\sharp)[G^\sharp]$ be an abstract aggregate value obtained by performing $s^\sharp$ parameterized with $e^\sharp$ over an abstract group $G^\sharp$. The abstract aggregate function $s^\sharp$ is sound if*

$$\forall G \in \gamma(G^\sharp), \quad s(e)[G] \in \gamma(s^\sharp(e^\sharp)[G^\sharp])$$

Example 2. Consider the database of Table 1(a) and the following query that computes the average ticket price for all flights departing after *7 o'clock* in the morning from any airport in Rome to any airport in Paris region:

Q_2 =`SELECT AVG(cost), COUNT(*) FROM flight WHERE source IN (FCO, CIA, LIRU, LIRV) and destination IN (ORY, CDG, BVA) and start-time` $\geq$ `7.00`

If we execute Q_2 on the database of Table 1(a), we get the result ξ_2 depicted in Table 3.

Table 3. ξ_2: Result of Q_2 (concrete)

$AVG(cost)$	$COUNT(*)$
260.44	2

The abstract version of Q_2 is defined as below:

$Q_2^\sharp$ =`SELECT`$^\sharp$ `AVG`$^\sharp$`(cost`$^\sharp$`), COUNT`$^\sharp$`(*) FROM flight`$^\sharp$ `WHERE source`$^\sharp$ `IN (Rome) and destination`$^\sharp$ `IN (Paris) and start-time`$^\sharp$ $\geq^\sharp$ `morning`

The result of $Q_2^\sharp$ on the abstract database of Table 1(b) is shown in Table 4. The

Table 4. $\xi_2^\sharp$: Result of $Q_2^\sharp$

$AVG^\sharp(cost^\sharp)$	$COUNT^\sharp(*)$
[0, 349.99]	[0, $+\infty$]

evaluation of the abstract WHERE clause extracts two tuple with $cost^\sharp$ equal to $[200.00, 249.99]$ and $[300.00, 349.99]$ both belonging to $G^\sharp_{may}$. So, $G^\sharp_{yes} = \{\}$. Thus, we get $a^\sharp = fn^\sharp(\{\}) = average^\sharp(\{\}) = [0, 0]$ and $b^\sharp = fn^\sharp(\{[200.00, 249.99], [300.00, 349.99]\}) = average^\sharp(\{[200.00, 249.99], [300.00, 349.99]\}) = [200.00, 349.99]$. Hence, $AVG^\sharp(cost^\sharp) = [min^\sharp(a^\sharp), max^\sharp(b^\sharp)] = [0, 349.99]$. In our example the abstraction of two or more concrete tuples may result into a single abstract tuple since the primary key is abstracted, so in such case $COUNT^\sharp(*) = [min^\sharp(a^\sharp), +\infty]$. observe that the result is sound *i.e.* $\xi_2 \in \gamma(\xi_2^\sharp)$.

Cooperative query evaluation with negation operation: Most of the existing abstraction-based techniques [4,5,6,13,15] suffer from the problem of soundness when query contains negation operation $MINUS$, because abstraction of the query appearing on right side of $MINUS$ may remove more information from the result of the query on left side of $MINUS$, yielding to a result that does not satisfy the soundness. In this section, we discuss the way to treat negation operation so as to preserve the soundness.

Given any abstract query $Q^\sharp$ and an abstract database state $dB^\sharp$, the result of the query can be denoted by $\xi^\sharp = [\![Q^\sharp]\!](dB^\sharp) = (\xi^\sharp_{yes}, \xi^\sharp_{may})$ where $\xi^\sharp_{yes}$ is the part of the query result for which semantic structure of $\phi^\sharp$ evaluates to true and $\xi^\sharp_{may}$ represents the remaining part for which $\phi^\sharp$ evaluates to $\top$[1]. Observe that we assume $\xi^\sharp_{yes} \cap \xi^\sharp_{may} = \emptyset$.

Consider an abstract query of the form $Q^\sharp = Q^\sharp_l \; MINUS^\sharp \; Q^\sharp_r$. Let the result of $Q^\sharp_l$ and $Q^\sharp_r$ be $\xi^\sharp_l = (\xi^\sharp_{yes_l}, \xi^\sharp_{may_l})$ and $\xi^\sharp_r = (\xi^\sharp_{yes_r}, \xi^\sharp_{may_r})$ respectively. The difference operation $MINUS^\sharp$ over an abstract domain is defined as follows:

$$\begin{aligned} \xi^\sharp &= \xi^\sharp_l \; MINUS^\sharp \; \xi^\sharp_r = \langle \xi^\sharp_{yes_l}, \xi^\sharp_{may_l} \rangle \; MINUS^\sharp \; \langle \xi^\sharp_{yes_r}, \xi^\sharp_{may_r} \rangle \\ &= \langle \xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r}), (\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r} \rangle \end{aligned} \tag{1}$$

Observe that the first component $(\xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r}))$ contains those tuples for which the pre-condition $\phi^\sharp$ strictly evaluates to *true*, whereas for the second component $((\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r})$ it evaluates to $\top$.

Example 3. Consider the database of Table 1(a) and the following query that finds all flights with ticket price strictly less than *205 USD*:

$Q_3 = Q_l$ `MINUS` Q_r, where Q_l = `SELECT * FROM flight`; and Q_r = `SELECT * FROM flight WHERE cost` $\geq$ 205.00;

The execution of Q_3 on the database of Table 1(a) yields the result ξ_3 shown in Table 5.

Table 5. ξ_3: Result of Q_3 (concrete)

flight no.	source	destination	cost ($)	start-time	reach-time	availability
$F004$	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
$F005$	Treviso (TSF)	Granby-Grand County (GNB)	200.15	16.00	17.20	Y

The corresponding abstract version of Q_3 is as follows:

$Q^\sharp_3 = Q^\sharp_l$ `MINUS`$^\sharp$ $Q^\sharp_r$, where $Q^\sharp_l$ = `SELECT`$^\sharp$ `* FROM flight`$^\sharp$; and $Q^\sharp_r$ = `SELECT`$^\sharp$ `* FROM flight`$^\sharp$ `WHERE cost`$^\sharp$ $\geq^\sharp$ [200.00, 249.99];

By following the abstract computation of $MINUS^\sharp$ defined in Equation 1, we get the result of $Q^\sharp_3$ depicted in Table 6 which is sound *i.e.* $\xi_3 \in \gamma(\xi^\sharp_3)$.

[1] When abstract query $Q^\sharp$ uses aggregate functions $s^\sharp$, application of $s^\sharp$ over a group $G^\sharp$ yields to a single row in $\xi^\sharp$. This row belongs to $\xi^\sharp_{may}$ only if all rows of that group belong to $G^\sharp_{may}$, otherwise it belongs to $\xi^\sharp_{yes}$.

Table 6. $\xi_3^\sharp$: Result by performing abstract computation of $Q_3^\sharp$

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Rome	Lyon	[100.00-149.99]	night	night	$\top$
$\top$	Venice	Lyon	[200.00-249.99]	afternoon	evening	$\top$

4.3 Concretization of the Cooperative Abstract Result

Given a concrete and the corresponding abstract databases dB and $dB^\sharp$ respectively, let $\xi^\sharp$ be the abstract answer obtained by executing the abstract query $Q^\sharp$ on $dB^\sharp$. The cooperative answer $R = \mathfrak{B}(dB, C, Q)$ returned to the user is obtained by: $R = \gamma(\xi^\sharp) \cap dB$. That is, the cooperative answer is obtained by mapping the abstract result into its concrete counterpart. For instance, after mapping $\xi_1^\sharp$ (Table 2), the user gets its concrete counterpart R_1 shown in Table 7.

Table 7. R_1: Concrete Result obtained by concretizing the abstract result $\xi_1^\sharp$

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

5 Soundness, Relevancy, and Optimality

Suppose, dB and $dB^\sharp$ represent a concrete database and its abstract version respectively. If Q and $Q^\sharp$ are representing the queries on concrete and abstract domain respectively, let ξ and $\xi^\sharp$ be the results of applying Q and $Q^\sharp$ on the dB and $dB^\sharp$ respectively.

Definition 5. *Let $dB^\sharp$ be an abstract table and $Q^\sharp$ be an abstract query. $Q^\sharp$ is sound iff* $\forall dB \in \gamma(dB^\sharp).\ \ \forall Q \in \gamma(Q^\sharp) : Q(dB) \in \gamma(Q^\sharp(dB^\sharp))$.

Let us denote by the notation $\mathfrak{B} \blacktriangleleft D^{abs}$ the fact that the cooperative system $\mathfrak{B}$ uses the abstract domain D^{abs}, and by $D_1^{abs} \sqsupseteq D_2^{abs}$ the fact that the abstract domain D_1^{abs} is an abstraction of D_2^{abs}.

Definition 6. *Given two abstract domain D_1^{abs} and D_2^{abs}. The domain D_1^{abs} is an abstraction of D_2^{abs} (denoted $D_1^{abs} \sqsupseteq D_2^{abs}$) if $\forall X^\sharp \in D_2^{abs}$, $\forall x \in \gamma_2(X^\sharp)$, $\exists!\ l \in D_1^{abs} :\ \alpha_1(x) = l$, where α_1 is the abstraction function corresponding to D_1^{abs}, and γ_2 is the concretization functions corresponding to D_2^{abs}.*

The extensional query answering system uses zero level abstraction and it returns only the exact answer if available. More relaxation of the query indicates more abstraction used by the cooperative system, returning more cooperative information to the users. Thus whenever we tune the level of abstraction from lower to higher, the system returns monotonically increasing answer set, *i.e.*

If $(\mathfrak{B} \blacktriangleleft D_1^{abs})$ and $(\mathfrak{B}' \blacktriangleleft D_2^{abs})$ and $(D_1^{abs} \sqsupseteq D_2^{abs})$, then $\mathfrak{B}(dB, C, Q) \supseteq \mathfrak{B}'(dB, C, Q)$

When the term "relevancy" comes into the context, the tuning of abstraction must end at a particular point, after which the system returns irrelevant additional information that does not satisfy the constraints in $S(Q)$, where $S(Q)$ is the set of constraints that make explicit the answers relevant to the user. We call the abstraction used at that point as the best correct approximation of the database information. Best correct approximation, thus, depends on the context that defines $S(Q)$. More abstraction beyond the best correct approximation level makes the answer partially relevant as it includes additional irrelevant information *w.r.t.* $S(Q)$.

Example 4. The cooperative answer R_1 of the query Q_1 in Example 1 is shown in Table 7. Let the constraint set be $S(Q)$={flights must be destined in the airport ORY/BVA/CDG/LYS, flight duration must be less than 3 hours}. Observe that the cooperative answer in Table 7 is completely relevant as all tuples in the answer satisfy $S(Q)$. If we use an higher level of abstraction, for instance, if the source and destination airports in Table 1(a) are abstracted by the nations they belong (in our example, Italy and France), the corresponding cooperative answer R'_1 of the query Q_1 will contain all tuples of the concrete Table 1(a) except the tuple with flight no. F002. The answer R'_1 is partially relevant because one tuple among them (flight no. equal to F005) does not satisfy $S(Q)$.

Our system can work together with a filtering system that can filter out those tuples from the partially relevant results that do not satisfy $S(Q)$, and ranks the results based on the satisfiability of tuples *w.r.t.* $S(Q)$. However, the level of abstraction determines the efficiency of the system with respect to the processing time.

5.1 Partial Order between Cooperative Answers

Given a database dB, a query Q, and a context C, the cooperative system may return different cooperative answers to the user under context C depending on the level of abstraction of the abstract domain which is used. We define a partial order between any two cooperative answers: a cooperative answer $R = \mathfrak{B}(dB, C, Q)$ is said to be better than another answer $R' = \mathfrak{B}'(dB, C, Q)$ (denoted $R \leq R'$) if R is more optimal than R' (see definition 3). The partial-ordered set of all cooperative answers for a given query under given context forms a lattice. The bottom most element R_0 determines worst cooperative answer which is completely irrelevant *w.r.t.* $S(Q)$, whereas the top most element R_n is the best cooperative answer which is completely relevant *w.r.t.* $S(Q)$.

Example 5. The cooperative answer R_1 of the query Q_1 in Example 1 is shown in Table 7. When we abstract the airports in Table 1(a) by the nations they belong, the corresponding cooperative answer R'_1 of the query Q_1 will contain all tuples of the concrete Table 1(a) except the tuple with flight no. F002. Since R'_1 contains one irrelevant tuple (tuple with flight no. F005) *w.r.t.* $S(Q)$ as depicted in Example 4, after filtering out the irrelevant tuple we get the result R_2 shown in Table 8. Observe that R_2 is better than the result R_1 (*i.e.* $R_2 < R_1$) since R_2 is more optimal than R_1 according to definition 3.

Table 8. R_2: Cooperative result of Q_1 while using more abstraction

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F004	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

There exists a wide variety of abstract domains with different expressiveness and complexity that focus on the linear relationship among program variables, such as Interval Polyhedra [3,2] to infer interval linear relationship, or Difference-Bound Matrices [16] representing the constraints of the form $x - y \leq c$ and $\pm x \leq c$ where x, y are program variables and c is constant. We can exploit such abstract domains by focusing on the set of constraint $S(Q)$ that make the answers relevant to the users.

6 Conclusions

Our new cooperative query answering scheme needs to be further refined. In particular, we are currently investigating its application to more sophisticated scenarios on different abstract domains, in order to properly address the tradeoff between accuracy and efficiency.

Acknowledgement

Work partially supported by RAS project "TESLA - Tecniche di enforcement per la sicurezza dei linguaggi e delle applicazioni".

References

1. Braga, J.L., Laender, A.H.F., Ramos, C.V.: Cooperative relational database querying using multiple knowledge bases. In: Proceedings of the 12th International Florida Artificial Intelligence Research Society Conference, May 1999, pp. 95–99. AAAI Press, Orlando (May 1999)
2. Chen, L., Miné, A., Cousot, P.: A sound floating-point polyhedra abstract domain. In: Ramalingam, G. (ed.) APLAS 2008. LNCS, vol. 5356, pp. 3–18. Springer, Heidelberg (2008)
3. Chen, L., Miné, A., Wang, J., Cousot, P.: Interval polyhedra: An abstract domain to infer interval linear relationships. In: Palsberg, J., Su, Z. (eds.) SAS 2009. LNCS, vol. 5673, pp. 309–325. Springer, Heidelberg (2009)
4. Chu, W.W., Chen, Q.: A structured approach for cooperative query answering. IEEE Transactions on Knowledge and Data Engineering 6(5), 738–749 (1994)
5. Chu, W.W., Chen, Q.: Neighborhood and associative query answering. Journal of Intelligent Information Systems 1(3-4), 355–382 (1992)

6. Chu, W.W., Yang, H., Chiang, K., Minock, M., Chow, G., Larson, C.: Cobase: a scalable and extensible cooperative information system. Journal of Intelligent Information Systems 6(2-3), 223–259 (1996)
7. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 238–252. ACM Press, New York (1977)
8. Gaasterland, T.: Cooperative answering through controlled query relaxation. IEEE Expert: Intelligent Systems and Their Applications 12(5), 48–59 (1997)
9. Giacobazzi, R., Ranzato, F., Scozzari, F.: Making abstract interpretations complete. Journal of the ACM (JACM) 47(2), 361–416 (2000)
10. Hachani, N., Ounelli, H.: A knowledge-based approach for database flexible querying. In: DEXA 2006, pp. 420–424. IEEE CS, Krakow (2006)
11. Halder, R., Cortesi, A.: Abstract interpretation for sound approximation of database query languages. In: Proceedings of the IEEE 7th International Conference on INFOrmatics and Systems (INFOS 2010), Advances in Data Engineering and Management Track, March 28–30, pp. 53–59. IEEE Catalog Number: IEEE CFP1006J-CDR, Cairo (2010)
12. Halder, R., Cortesi, A.: Observation-based fine grained access control for relational databases. In: Proceedings of the 5th International Conference on Software and Data Technologies (ICSOFT 2010), July 22–24, pp. 254–265. INSTICC Press, Athens (2010)
13. Huh, S.-Y., Moon, K.-H.: Approximate query answering approach based on data abstraction and fuzzy relation. In: Proceedings of INFORMS-KORMS, Seoul, Korea, pp. 2057–2065 (June 2000)
14. Ichikawa, T., Hirakawa, M.: Ares: a relational database with the capability of performing flexible interpretation of queries. IEEE Transactions on Software Engineering 12(5), 624–634 (1986)
15. Keun Shin, M., Huh, S.Y., Lee, W.: Providing ranked cooperative query answers using the metricized knowledge abstraction hierarchy. Expert Systems with Applications 32(2), 469–484 (2007)
16. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Danvy, O., Filinski, A. (eds.) PADO 2001. LNCS, vol. 2053, pp. 155–172. Springer, Heidelberg (2001)
17. Palpanas, T., Koudas, N.: Entropy based approximate querying and exploration of datacubes. In: Proceedings of the 13th International Conference on Scientific and Statistical Database Management (SSDBM 2001), pp. 81–90. IEEE Computer Society Press, Fairfax (2001)