

# How to recognize interesting topics to provide Cooperative Answering \*

F.Cuppens            R.Demolombe

ONERA-CERT  
2 Av. E.Belin, BP3025  
Toulouse Cedex  
France

June 12, 2002

## Abstract

We present a method, and its formalization, to provide interesting additional information to users asking queries to a Relational Data Base. The interesting information is defined using a Knowledge Base containing rules which represent the expertise of an expert having a long experience in providing information to casual users.

This Knowledge Base also contains a high level description of the Data Base content. This description uses the concepts of entity, attribute of entity, relationship and topic. The topics are related to attributes and relationships in order to be able to characterize all the information in the Data Base which belongs to a given semantic field.

The Data Base and Knowledge Base are both formalized in First Order Logic. There are two different formalization levels: object level to represent the Data Base itself, and meta level to represent the Knowledge Base which is used to transform queries. The transformed queries define the additional information which is provided to users. They can be obtained with a standard deduction mechanism.

A first prototype implemented in Prolog is running.

## 1 Introduction

In [5], we have presented a general methodology to provide intelligent access to Data Bases, which is called Cooperative Answering. The basic idea is to help users who have to retrieve data in contexts where the data to be retrieved are not precisely defined, such as Decision Support Systems, or Advice Giving Systems.

In these situations when a user asks a query, for example a Relational query, the answer is precisely defined, and a Relational DBMS can provide the exact answer. However in most cases it may happen that the user

---

\*This work was supported by the ESPRIT project: ESTEAM-316

might be interested by other data, which are related to his query, but are not explicitly requested.

The Cooperative Answering approach is the following. If a system has to determine this additional interesting information, it has to determine what are the user's topics of interest, and then to determine what relevant data in the Data Base, corresponding to these topics, must be retrieved.

We present examples below in order to illustrate the Cooperative Answering approach. Let's consider a person who wants to organize a travel from Paris to New-York and asks the question:

*What is the departure time of the flights from Paris to New-York, whose departure time is between 7a.m. and 11a.m. ?*

The answer provided by a standard Relational DBMS is, for example:

Flight	Departure-time
AF001	10h00
AF001	11h00
AF015	10h30

The answer provided by a person, working in a travel agency, and having a long experience in giving advices to his clients, would be, for example:

Flight	Departure-time	Arrival-time	Validity	Works-on	Price
AF001	10h00	08h45	01/01 27/09		27,715 FF
AF001	11h00	09h45	28/09 31/12		27,715 FF
AF015	10h30	14h55	01/01 27/09	1,3,6	
AF015	10h30	14h55	28/09 31/12	1,3,6,7	

This is not the exact answer, since it provides additional information, like the arrival-time. This information can be interesting for a person who is organizing a travel, for example if he has to take a connection. The validity period is also interesting because it allows to understand why the flight AF001 has two different departure times, each one corresponding to a different validity period. On the other hand, in the case where a flight is not available all days in the week, it is useful to show which days it works, because people assume that a flight is usually available all days in the week. Finally, in the case where a flight is very expensive, it is interesting to tell to the client what is the price, even if he is not specially motivated by the price because he is travelling for business. In our example the aircraft corresponding to the flight AF001 is the Concorde, and its price is roughly three times the usual price; such a price is very exceptional, that is why it is useful to inform the client of this fact.

The previous example has shown the style of answers which could be expected from a system which tries to be as cooperative as possible. We can induce from it the type of query transformation a system has to realize to provide this kind of additional information.

In [5], we have explicitated three kinds of additional information a system can provide to a user:

- Additional entities having a different type than those requested in the query.
- Additional entities satisfying “neighbourhood” conditions.
- Additional attribute values.

In this paper we focus on the third one, and we propose a more detailed, and more formal presentation of how to provide additional attribute values.

In this case, if answers are considered as tableaux representing the exact answers, the extensions to these answers can be viewed as an extension of the tableaux in the width. The elements of the tableaux without any value correspond to attribute values which are not explicitly represented and which are not interesting. They do not correspond to any kind of “null value”. One can also remark that an extended answer might be represented by several tableaux having different structures.

To design this method we have defined a query transformation: the result of this transformation is a query whose answer contains the additional information. The transformation uses different kinds of knowledge which are presented in the next sections. They correspond to: a high level representation of the Data Base content in terms of the Entity-Relationship concepts, an Entity type structure, topics which are related to attributes or relationships, a topic structure, a query representation, and a user representation. All this knowledge is formalized by facts or rules in First Order Logic. Moreover, there are general rules defining how to use the previous knowledge to transform the queries; they represent expertise in Cooperative Answering, in general. And rules which are specific to a given application domain. The overall knowledge is considered as a Theory, and transformed queries are obtained through by application of standard inference rules.

## 2 High level representation of the Data Base content

Our intent is to represent the overall knowledge in the formalism of First Order Logic (FOL). However the concepts embedded in FOL are very simple, and easy to understand, but their semantics is poor and they don't allow to represent, in a straightforward way, concepts having richer semantics, such as those of the Entity-Relationship model.

One way to solve this problem is to distinguish, as suggested by R.Kowalski in [3] and [7], two different representation levels: Object level and Meta level.

- The Object level, in which is represented the Data Base and the queries; this Data Base can contain rules, if we are in a Deductive Data Base context, but these rules must not be confused with the rules in the Knowledge Base defined in the previous section; at this level all predicates have the same status, and there is no difference, for instance, between relationships and attributes.

- The Meta level, in which is represented the semantics which cannot be represented at the Object level; in particular the Knowledge Base and the query transformation are represented at the Meta level. We also use this Meta level to have a high level representation of the data base content.

In the next sections, we give an explicit correspondence between the two representation levels of data base content, and theoretical results show the completeness and soundness of this correspondence (see section 2.3).

## 2.1 Object level and Meta level formalization

The Object language is a First Order Language whose predicates are used to represent the facts stored in the Data Base.

At the Meta level, facts or queries of the Object level, are considered as concrete objects represented by constants or terms. The correspondence between the two levels is defined by a coding function which assigns a code to each formula of the Object level [3]. For example, we might code the following atomic formula :

$$\begin{aligned} & \textit{Departure-city}(x, \textit{Paris}) \\ & \textit{Arrival-city}(x, \textit{New-York}) \end{aligned}$$

by the terms:

$$\begin{aligned} t1: & \textit{atom}(\textit{pred}(\textit{Departure-city}), \textit{var}(x), \textit{const}(\textit{Paris})) \\ t2: & \textit{atom}(\textit{pred}(\textit{Arrival-city}), \textit{var}(x), \textit{const}(\textit{New-York})) \end{aligned}$$

where *atom*, *pred*, *atom*, *var* et *const* are function symbols.

The conjunction of these two formulas is represented by the term:

$$\textit{and}(t1, t2)$$

In the following such terms are abbreviated by the Object formula between quotes. For example the previous term is abbreviated by:

$$\textit{“Departure-city}(x, \textit{Paris}) \wedge \textit{Arrival-city}(x, \textit{New-York)}\textit{”}$$

Such terms can contain Meta variables to represent an Object formula which is not completely explicited. In that case, we add unquoting marks  $\langle \rangle$  around all expressions which denote Meta variables. For example, in the expression :

$$\textit{“Departure-city}(\langle x \rangle, \textit{Paris})\textit{”}$$

*x* is a Meta variable.

The predicates of the Meta language are presented in the following subsections, and to sum up, a list of the meta predicates used in this paper is given in appendix. The Meta language is a First Order language built with these predicates.

## 2.2 Entity-Relationship model representation

In this section is represented an Entity-Relationship model formalization using object and meta levels.

### 2.2.1 Types

In this model all the elements have a type, and there are two kinds of types: the Entity types, and the Attribute value types.

At the object level each type is represented by a monadic predicate, at the meta level they are represented by constants (the code of these monadic predicates). At the meta level Entity types, and Attribute value types are distinguished through the meta predicates:

- $Entity\text{-}type(x)$
- $Attribute\text{-}value\text{-}type(x)$

Example:

Object level:

- $Flight(x), Hour(x), City(x)$

Meta level:

- $Entity\text{-}type("Flight")$
- $Attribute\text{-}value\text{-}type("Hour")$
- $Attribute\text{-}value\text{-}type("City")$

In the following, when there is no risk of ambiguity, the quotes representing codes at the meta level are omitted.

### 2.2.2 Predicate arguments typing

At the object level, there are axioms to define the type of each predicate argument. At the meta level, this information is represented with the meta predicate:

- $Type(x,y)$

where  $x$  is a predicate code, and  $y$  is a list of type codes, each type in the list corresponding to the argument having the same rank.

Example:

Object level:

- $\forall(x,y) Departure\text{-}time(x,y) \longrightarrow Flight(x) \wedge Hour(y)$
- $\forall(x,y) Reservation(x,y) \longrightarrow Flight(x) \wedge Person(y)$
- $\forall(x,y,z) Seat\text{-}numb(x,y,z) \longrightarrow Flight(x) \wedge Person(y) \wedge Seat(z)$

Using the Entity-Relationship model concepts, the predicate *Departure-time* represents an attribute of entity, the predicate *Reservation* represents a relationship, and the predicate *Seat-numb* represents an attribute of relationship.

Meta level:

- $Type(Departure-time, [Flight, Hour])$
- $Type(Reservation, [Flight, Person])$
- $Type(Seat-numb, [Flight, Person, Seat])$

where  $[Flight, Hour]$  is a shorthand for the term which represents the list: Flight, Hour.

### 2.2.3 Attribute of entities

For each Entity type, and for each attribute which is defined for an Entity type, there is an axiom at the object level expressing that any entity of this type has a value for this attribute. There is the same kind of axioms for attributes of relationship. At the meta level this information is represented with the predicate:

- $Att(x, y)$

where  $x$  is an attribute and  $y$  is an entity type, or a relationship.

Example:

Object level:

- $\forall(x) Flight(x) \longrightarrow \exists(y) Departure-time(x, y)$
- $\forall(x, y) Reservation(x, y) \longrightarrow \exists(z) Seat-numb(x, y, z)$

Meta level:

- $Att(Departure-time, Flight)$
- $Att(Seat-numb, Reservation)$

These facts express that *Departure-time* is an attribute of the Entity type *Flight*, and that *Seat-numb* is an attribute of the relationship *Reservation*.

## 2.3 Entity type structure

To have a more structured view of the Entity types there is an Entity type hierarchy. At the object level there are axioms to represent that two Entity types are included or disjointed. The same information is represented at the meta level with the predicates:

- $ISA(x, y)$
- $DIS(x, y)$

where  $x$  and  $y$  are Entity types.

Example:

Object level:

The following axioms express that Flight and Train are both Travel, that Flight and Train are disjoint, and that Travel and Person are disjoint.

- $\forall(x) \text{ Flight}(x) \longrightarrow \text{Travel}(x)$
- $\forall(x) \text{ Train}(x) \longrightarrow \text{Travel}(x)$
- $\neg(\exists(x) \text{ Flight}(x) \wedge \text{Train}(x))$
- $\neg(\exists(x) \text{ Travel}(x) \wedge \text{Person}(x))$

Meta level:

- $\text{ISA}(\text{Flight}, \text{Travel})$
- $\text{ISA}(\text{Train}, \text{Travel})$
- $\text{DIS}(\text{Flight}, \text{Train})$
- $\text{DIS}(\text{Travel}, \text{Person})$

To reflect all the semantics of the ISA and DIS predicates, we have to insert in the meta theory the following axioms, which express the reflexivity and transitivity of the ISA predicate, the symmetry of the DIS predicate, and the link between these two predicates:

- $R1: \forall(e) \text{ Entity-type}(e) \longrightarrow \text{ISA}(e, e)$   
 $R2: \forall(e, e', e'') \text{ ISA}(e, e') \wedge \text{ISA}(e', e'') \longrightarrow \text{ISA}(e, e'')$   
 $R3: \forall(e, e') \text{ DIS}(e, e') \longrightarrow \text{DIS}(e', e)$   
 $R4: \forall(e, e', e'') \text{ DIS}(e, e'') \wedge \text{ISA}(e', e'') \longrightarrow \text{DIS}(e, e')$

It is not too much complex to prove that the set of axioms  $S = \{R1, R2, R3, R4\}$ , of the meta theory reflects the intended meaning of the ISA and DIS predicates. That is, these axioms are sound and complete in the following sense:

Completeness: let's consider a set of axioms *Obj-ax* at the object level of the form:  $(T(x) \longrightarrow T'(x))$ , or  $\neg(\exists(x) T(x) \wedge T'(x))$ , and the set of corresponding axioms *Meta-ax*, at the meta level, of the form:  $\text{ISA}(T, T')$  or  $\text{DIS}(T, T')$ . Moreover, let  $F$  be a set of facts *Entity-type*( $T_1$ ) ... *Entity-type*( $T_n$ ) where  $T_1 \dots T_n$  is the set of predicates which appear in *Obj-ax*. Then any theorem, at the object level, of the same form, which is derivable from *Obj-ax*, has a corresponding theorem at the meta level which is derivable from *Meta-ax*,  $S$  and  $F$ .

Soundness: each theorem of the form  $\text{ISA}(T, T')$  or  $\text{DIS}(T, T')$  derivable from *Meta-ax*,  $S$ , and  $F$  at the meta level, has a corresponding theorem, at the object level, of the form:  $(T(x) \longrightarrow T'(x))$  or  $\neg(\exists(x) T(x) \wedge T'(x))$ , derivable from *Obj-ax*.

We provide a proof of these results in the Appendix B.

If we add, at the object level, axioms of the form  $\exists(x)T(x)$ , for any entity type, to express entity types are not empty, then we have to add, at the meta level the axiom:

$$R5: \neg\exists(e, e')(ISA(e, e') \wedge DIS(e, e'))$$

With the axiom R5 the above results about completeness and soundness can be extended to theorems of the form:  $\neg ISA(T, T')$  and  $\neg DIS(T, T')$ .

To express inheritance of attribute from an entity type to a sub-type, we have also the following axiom:

$$R6: \forall(a, t, t') Att(a, t) \wedge ISA(t', t) \longrightarrow Att(a, t')$$

## 2.4 Topics

The topic concept has been introduced to relate predicates whose meanings are in the same semantic field. For instance the predicates *Departure-city* and *Arrival-city* can be related to the topic: *Location*. The link between a predicate and a topic cannot be directly expressed in First Order Logic, since it asserts some knowledge about a predicate, but by the separation into two distinct levels, predicates have the status of constant at the meta level, and it becomes possible to represent these assertions in First Order Logic.

The links between predicates represented through the topics is one of the ways which are used to derive what are the user's interesting topics. For instance if a person asks a query about a flight departure time, we can infer that this person may be interested by the arrival time, because they are both related to the topic: Time.

This concept is only defined at the meta level. It is represented by the predicates:

- *Topic(x)*
- *Att-Top(x,y)*

In *Att-Top(x,y)*, x is an attribute or a relationship, and y is a topic.

Example:

Meta level:

- *Topic(Time)*
- *Topic(Validity-condition)*
- *Topic(Time-table)*
- *Topic(Money)*

The previous facts define some topics.

- *Att-Top (Departure-time, Time-table)*

- *Att-Top (Arrival-time, Time-table)*
- *Att-Top (Price, Money)*
- *Att-Top (Works-on, Validity-condition)*
- *Att-Top (Validity, Validity-condition)*

The previous facts express, for instance, that the attribute *Works-on* is related to the topic *Validity-condition*.

The topics are structured with the same predicate ISA as Entity types. We have, for example:

- *ISA (Time-table, Time)*
- *ISA (Validity-condition, Time)*

### 3 Query language

Queries are expressed at the object level. Their semantics is represented in First Order Predicate Calculus. However the form of the queries is strongly restricted, from a syntactical point of view, though this form doesn't restrict strongly the expressive power of Predicate Calculus. Roughly speaking any query expressed in the language Alpha, introduced by E.F.Codd in [4], can be expressed in the form defined below.

This form corresponds to the query structure presented in the introduction:

$$Entity \wedge Condition \wedge Retrieved\ Attributes$$

where:

“*Entity*” is a formula with the only logical operators  $\wedge$  (conjunction) or  $\vee$  (disjunction), and where predicates are only monadic predicates corresponding to Entity types. This part of the query defines the type of entities which must be retrieved.

“*Condition*” is any formula of the object language. This part defines the conditions these entities have to satisfy.

“*Retrieved Attributes*” is a formula with only  $\wedge$  logical operator and where the predicates are only predicates corresponding to Attributes. This part defines the attribute values the user wants to know.

Moreover the following syntactical constraints are imposed to the form of the queries:

- any free variable of the Condition part must be a free variable of the Entity part or of the Retrieved Attributes part.
- queries must be Domain Independent formulas, in the sense defined in [6].

Queries are represented at the meta level with the predicates:

- *Query (x)*: x is a query.
- *Entity (x,y), Condition (x,y), Ret-Att (x,y)*: y is the code of a formula which respectively represents the Entity part, the Condition part and the retrieved attributes part of a query x.

We have also to notice that the additional information provided in the Cooperative Answering approach cannot be represented in general with a unique formula. Indeed, as we have shown in the example of the introduction, this information should be represented by several “tableaux” having different structures, in particular different attribute number, which correspond in Logic to formulas having different free variable number. Then one way to represent the query transformation result would be a set of queries. However that would lead to the idea to have an answer which is a set of “heterogeneous tableaux”, which is not so natural than to have a unique tableau where the number of attributes is not the same for each row. This latter representation is more natural since it carries on the intuitive idea that, depending on the value of some attributes in a given row, some other additional attribute values must be provided or not for this row.

That is the reason why, though from a theoretical point of view the transformation result is a set of queries, we have defined a specific logical operator, called pseudo-imply, denoted by  $\implies$ , which allows to represent this set of queries with a unique query. This operator is not just a trick to have a more concise representation, it also allows an easier definition of transformation rules as we will see latter.

This can be made clearer with a simple example. Let’s consider the query qt1:

*qt1:*  
*Entity: Flight(x)*

and let’s assume that, applying an expert rule, we have to generate a new query which provides the price of flights, only for those which are very expensive. Let’s *Expensive(x)* be a formula defining the very expensive flights. Then, in our formalism, the transformation result is:

$$Flight(x) \wedge (Expensive(x) \implies Price(x, y))$$

The intuitive meaning of this formula is that if, for a given x value, *Expensive(x)* is false, the formula has only one free variable, which is x, and if *Expensive(x)* is true, the formula has two free variables, which are x and y. That is the y value is provided only for expensive flights.

That could be also represented by the two queries:

*qt2: Flight(x)  $\wedge$   $\neg$ Expensive(x)*  
*qt3: Flight(x)  $\wedge$  Expensive(x)  $\wedge$  Price(x,y)*

A more formal definition of the pseudo-imply operator can be found in [5]. In the following, the queries containing the  $\implies$  operator are called “heterogeneous queries”.

The formal definition of the answers is defined as usual (see for example [8]). For heterogeneous queries the answer, is formally defined as the set of answers to the queries represented by the unique heterogeneous query. In [5] we have shown that it is more meaningful to present these answers in the form of instantiated formulas than in the form of tuples. In the example in the Annex answers are represented in this form.

## 4 Expert's knowledge representation

In this section is presented the knowledge which allows the system to simulate the behaviour of a person who has experience in answering questions for users who have to design a plan or to take a decision. For instance this person may be a salesman in a travel agency.

This knowledge contains a user's representation, because the interest of some piece of information strongly depends on the user's characteristics and believes. It also contains general rules to derive what are the user's topics of interest, and general rules to derive what are the attribute for which it is relevant to provide the values.

Some of this rules express general expertise in Cooperative Answering, others depend on the application domain.

### 4.1 User representation

The user representation can be more and more sophisticated. In this section we just want to show how it can be expressed in our formalism.

The simplest way to represent a user is to have a set of user's types, and to tell what is the type of a given user. For example if we find useful to know if a user is traveling for touristic reasons, or for business, we can introduce the two corresponding user's types.

These types are represented at the meta level with the meta predicate:

- $User\text{-}type(x)$

and we may have, at the meta level:

- $User\text{-}type(Tourist)$
- $User\text{-}type(Businessman)$

We can also represent common sense believes which are assumed by any user.

For example, if no fact contradicts them, all users will assume the following rules:

- A flight is available from January 1st to December 31st.
- A flight is available all days of the week.
- A flight is not a surcharge flight.

These believes could be represented at the object level by the rules:

$$D1: \forall (x) \\ \text{Flight}(x) \longrightarrow \text{Validity}(x,01/10,31/12)$$

$$D2: \forall (x, y) \\ \text{Flight}(x) \wedge \text{Day}(y) \longrightarrow \text{Works-on}(x, y)$$

$$D3: \forall (x) \\ \text{Flight}(x) \longrightarrow \neg \text{Surcharge}(x)$$

These rules are assumed “by default”, but users know that they can have exceptions. The facts which express these exceptions are interesting facts and must be provided to the user.

At the meta level, to represent these knowledge, we introduce in the language the predicate:

*Normal-Belief*(“ $E(x)$ ”, “ $a$ ”, “ $C(x)$ ”): For any entity  $x$  of type  $E$ , all users assume that the condition  $C(x)$  holds; if it is not the case, the value of the attribute  $a$  has to be provided for the entity  $x$ .

Example:

$$RD1: \text{Normal-Belief}(\text{“Flight}(x)\text{”}, \text{“Validity”}, \text{“Validity}(x,01/01,31/12)\text{”})$$

$$RD2: \text{Normal-Belief}(\text{“Flight}(x)\text{”}, \text{“Works-on”}, \\ \forall (y) \text{Day}(y) \longrightarrow \text{Works-on}(x, y)\text{”})$$

$$RD3: \text{Normal-Belief}(\text{“Flight}(x)\text{”}, \text{“Price”}, \text{“}\neg \text{Surcharge}(x)\text{”})$$

are respectively the translation at the meta level of the object rules D1, D2 and D3.

## 4.2 Rules to recognize interesting topics

The rules R7 and R8 are independent of the application domain. The rule R7 expresses that the topics related to attributes which appear in the query are interesting topics for this query:

$$R7: \forall (q, a, top) \\ (\text{Appear}(q, a) \wedge \text{Att-Top}(a, top)) \longrightarrow \text{Int-Top}(q, top)$$

where the meta predicates:

- $\text{Appear}(x, y)$
- $\text{Int-top}(x, y)$

mean that the attribute  $y$  appears in the query  $x$ , and the topic  $y$  is interesting for the query  $x$ . The predicate  $\text{Appear}$  is defined by several rules which decompose the condition part and the retrieved attribute part of the query.

The rule R8 expresses that in the topic structure the concept of interest is inherited:

$$R8: \forall (q, top, top') \\ (Int-Top (q, top) \wedge ISA (top', top)) \longrightarrow Int-Top (q, top')$$

The following rule RD4 is only valid in the application domain of a travel agency. It says that the topic Money is an interesting one for any query asked by users who are travelling for tourism:

$$RD4: \forall (q) \\ Query (q) \wedge User-type (Tourist) \longrightarrow Int-Top (q, Money)$$

### 4.3 Rules to derive relevant attributes

The rule R9 allows to derive the relevant attributes from the topics which have been recognized to be interesting. The rule says that if there is in the Entity part of the query an entity  $v$  of type  $e$ , which has an attribute  $a$  related to an interesting topic then it is relevant to provide in addition the value of this attribute for the entity  $v$ . This value will be the value of the object variable  $v_1$ , whose code is a constant at the meta level.

$$R9: \forall (q, e, x, top, a) \\ (entity (q, "e(< x >)") \wedge \\ Int-Top (q, top) \wedge \\ Att (a, e) \wedge \\ Att-Top (a, top)) \\ \longrightarrow Rel-Att (q, "e(< x >)", "a(< x >, v_1)")$$

where  $Rel-Att (q, "e(< x >)", "a(< x >, v_1)")$  means that it is relevant to provide the value of the attribute  $a$  for the entity  $x$  of type  $e$ , which appears in the Entity part of the query  $q$ .

As object predicates have different number of arguments, it is important to notice that  $v_1$  actually represents a tuple of object variable codes.

If several attributes, say  $a_1$  and  $a_2$ , of the same entity type  $e$  are relevant, the atomic formulas  $a_1(x, v_1)$  and  $a_2(x, v_1)$  might be added in the retrieved attribute part of the query. This would lead to introduce a link, through the object variable  $v_1$ , which is completely artificial. To avoid this situation it is necessary to change the variable names when the retrieved attribute part is generated. For example, in this case,  $a_2(x, v_2)$  is generated, instead of  $a_2(x, v_1)$ .

We also use the following rules:

$$R10: \forall (q, E, x_1, x_2, a, C, C') \\ entity(q, "E(< x_1 >)") \wedge \\ Normal-Belief("E(< x_2 >)", a, C) \wedge \\ Substitute(C, x_2, x_1, C') \\ \longrightarrow Rel-Att(q, "E(< x_1 >)", "(\neg C') \implies a(< x_1 >, v_1)")$$

The rule R10 expresses that if there is a query about entities  $x_1$  of type  $E$  and there is a fact, like RD1, RD2 or RD3, representing normal believes

$C$  about this type of entities, then the value of the attribute  $a$  must be provided for the entities which do not satisfy  $C$ ; that is for entities which are exceptions. The pseudo-imply, in *Rel-Att*, allows to represent these exceptional situations.

The *Substitute* predicate change the variable name  $x$  in  $C(x)$  in *Normal-Belief*, into the corresponding variable name in the query. Intuitively, we have to express that the condition  $C$  applies to entities satisfying the query.

#### 4.4 General rules to transform a query

The general rules defining the transformed queries are listed below.

We have first a set of rules R11, R12 and R13 whose role is to derive the set of atomic formulas defining the entity types which appear in the Entity part of the query:

$$R11: \forall (q, e) \\ Entity(q, e) \longrightarrow atom-entity(q, e)$$

$$R12: \forall (q, e_1, e_2) \\ atom-entity(q, "e_1 \wedge e_2") \longrightarrow atom-entity(q, e_1) \wedge atom-entity(q, e_2)$$

$$R13: \forall (q, e_1, e_2) \\ atom-entity(q, "e_1 \vee e_2") \longrightarrow atom-entity(q, e_1) \wedge atom-entity(q, e_2)$$

where the meta predicate  $atom-entity(x, y)$  means that in the Entity part of the query  $x$  appears the atomic formula  $y$ .

The rule R14 expresses that if there is an entity type which is not disjointed with an entity type appearing in the query, then, by extension, this entity type is considered as an entity type of the query.

$$R14: \forall (q, e, e', x) \\ Entity-type(e') \wedge atom-entity(q, "e(< x >)") \wedge \neg DIS(e, e') \\ \longrightarrow entity(q, "e'(< x >)")$$

where the meta predicate  $entity(x, y)$  means that the entity type  $y$  is not disjointed with some entity type in the Entity part of the query  $x$ .

The reason why we have introduced the predicate  $entity$  is that it may happen that, in a query an atomic formula appears in the Entity part, for example  $man(x)$ , and that it is possible to derive for another entity type, for example  $teacher(x)$ , which is not disjointed with  $man(x)$  and doesn't appear in the query, that it is relevant to provide the value of some attribute  $a$  for teachers, but not for any man. In this situation the formula:  $teacher(x) \implies a(x, y)$  is added to the list of attributes to be retrieved, though  $teacher(x)$  is not in the query.

The formulas DEF1 and DEF2 define two meta predicates used in the rules R15 and R16.

The meta predicate *Rel-Att-for-all*( $q, f$ ) means that it is possible to derive for all entity types in the Entity part of the query that the attribute represented by the atomic formula  $f$  is relevant for these entity types.

The meta predicate  $Rel-Att-for-some(q, "e(< v >)", f)$  means that the attribute represented by  $f$  is relevant at least for the entity type  $e$ , which is not disjoint with some entity type in the query  $q$ .

$$\begin{aligned}
 DEF1: \forall (q, f) \\
 & Rel-Att-for-all(q, f) \longleftrightarrow \\
 & \quad \forall (e, v) \exists (e') (atom-entity(q, "e(< v >)" ) \\
 & \quad \quad \rightarrow Rel-Att(q, "e'(< v >)", f) \wedge ISA(e, e'))
 \end{aligned}$$

$$\begin{aligned}
 DEF2: \forall (q, e, v, f) \\
 & Rel-Att-for-some(q, "e(< v >)", f) \longleftrightarrow \\
 & \quad entity(q, "e(< v >)" ) \wedge Rel-Att(q, "e(< v >)", f)
 \end{aligned}$$

The rule R15 expresses that if an attribute  $f$  is relevant for any entity type in the query, then it must be added in the list of attributes to be retrieved.

$$\begin{aligned}
 R15: \forall (q, f) \\
 & Rel-Att-for-all(q, f) \rightarrow Add-Att(q, f)
 \end{aligned}$$

The rule R16 expresses that, if an attribute  $f$  is relevant for some entity type, but not for all entity types, then the attribute value of  $f$  has to be provided additionally, only for entities of type  $e$ . That is why the formula added in the list of attributes to be retrieved contains a pseudo-imply.

$$\begin{aligned}
 R16: \forall (q, e, v, f) \\
 & Rel-Att-for-some(q, "e(< v >)", f) \wedge \neg Rel-Att-for-all(q, f) \\
 & \quad \rightarrow Add-Att(q, "e(< v >)" \Rightarrow f)
 \end{aligned}$$

## 5 Example of query transformation

We show below a short example of transformed queries obtained by the derivation process. Only the most important steps of the derivation are exhibited.

Let's consider the following query:

*Query (qt1)*

*Entity (qt1, "Flight(x)")*

*Condition (qt1, "Departure-city(x, Paris)  $\wedge$   
Arrival-City(x, New-York)  $\wedge$   
Departure-time(x, y)  $\wedge$  (8 < y)  $\wedge$  (y < 12)")*

*Ret-Att (qt1, "Departure-time(x, y)")*

We have: *Entity (qt1, "Flight(x)")*

then from R11, we get: *atom-entity(qt1, "Flight(x)")*

From R1, we get: *ISA (Flight, Flight)*

then from R5, we get:  *$\neg DIS$  (Flight, Flight)*

and by using R14, we get: *entity (qt1, "Flight(x)")*

We have:

*Appear(qt1, Departure-time)*

*Att-Top(Departure-time, Time-table)*

then, from R7, we get: *Int-Top(qt1, Time-table)*

We have:

*Att(Arrival-time, Travel)*

*ISA(Flight, Travel)*

then, from R6, we get: *Att(Arrival-time, Flight)*

We have: *Att-Top(Arrival-time, Time-table)*

then from R9, we get:

*Rel-Att(qt1, "Flight(x)", "Arrival-time(x, v<sub>1</sub>)")*

From RD1, RD2, RD3 and from R10, we get:

*Rel-Att(qt1, "Flight(x)", " $\neg$  Validity(x, 01/01, 31/12)  $\implies$  Validity(x, v<sub>1</sub>, v'<sub>1</sub>)")*

*Rel-Att(qt1, "Flight(x)", " $\neg$ ( $\forall$ (y) Day(y)  $\longrightarrow$  Works-on(x, y))  
 $\implies$  Works-on(x, v<sub>1</sub>)")*

*Rel-Att(qt1, "Flight(x)", " $\neg\neg$  Surcharge(x)  $\implies$  Price (x, v<sub>1</sub>)")*

From DEF1 and R15, and after formula simplification, we get:

*Add-Att(qt1, "Flight(x)", "Arrival-time(x, v<sub>1</sub>)")*

*Add-Att(qt1, "Flight(x)", "Surcharge(x)  $\implies$  Price (x, v<sub>1</sub>)")*

*Add-Att(qt1, "Flight(x)", " $\exists$ (y)(Day(y)  $\wedge$   $\neg$  Works-on(x, y))  
 $\implies$  Works-on(x, v<sub>1</sub>)")*

*Add-Att(qt1, "Flight(x)", " $\neg$  Validity(x, 01/01, 31/12)  $\implies$  Validity(x, v<sub>1</sub>, v'<sub>1</sub>)")*

We have noticed in the previous section that when new atomic formulas are inserted in the list of attributes to be retrieved, it can happen that the same variable appears in several predicate arguments, and introduces artificial links. That is the case, for example for the variable  $v_1$  in *Departure-time(x, v<sub>1</sub>)*, *Price(x, v<sub>1</sub>)*, *Works-on(x, v<sub>1</sub>)* and *Validity(x, v<sub>1</sub>, v'<sub>1</sub>)*. To avoid that, the Rename predicate changes the variable name  $v_1$  in the above formulas into  $v_1$ ,  $v_2$ ,  $v_3$  and  $v_4$ .

Finally we get the transformed query qt2:

*Query (qt2)*

*Entity (qt2, "Flight (x)")*

*Condition (qt2, "Departure-city (x, Paris)  $\wedge$   
Arrival-City (x, New-York)  $\wedge$   
Departure-time (x, y)  $\wedge$  (8 < y)  $\wedge$  (y < 12)")*

*Ret-Att (qt2, "Departure-time (x, y)  $\wedge$   
Arrival-time (x, v<sub>1</sub>)  $\wedge$   
Surcharge (x)  $\implies$  Price (x, v<sub>2</sub>)  $\wedge$   
 $\exists$ (y)(Day (y)  $\wedge$   $\neg$  Works-on (x, y))  
 $\implies$  Works-on (x, v<sub>3</sub>)  $\wedge$*

$$\neg \text{Validity } (x, 01/01, 31/12) \implies \text{Validity } (x, v_4, v'_1)$$

## 6 Conclusion

We have presented a method, and its formalization, called Cooperative Answering, to provide additional interesting information when a user asks a Relational query to a Data Base.

General rules are used to represent the expertise of a person having a long experience in providing information to people who are not familiar with a given application domain. To determine what is the interesting information the method also uses a high level description of the Data Base content which is much more rich, from a semantic point of view, than a standard Relational schema. This description uses Entity Relationship model concepts, plus a structure of entity types, and the concept of topic.

The notion of interesting topic is very useful to define the interesting information. It offers a different and complementary approach to the approach followed by J.F.Allen in [1, 2], where the interesting information is defined using the notion of plan. A plan is a list of actions. The system knows a set of predefined plans, and if it recognizes, from user's queries the intention to realize some action in a given plan, then it provides to the user the additional information which can help him to realize the other actions in the plan. The plans allow a more precise definition of interesting information than topics, but there are many situations where the user has no precise plan in mind when he is asking a query, or where there is no predefined plan in the system corresponding to what he wants to do. In these cases the predefined plans cannot be used.

We have also shown how the representation of some user's normal believes allows to define another kind of interesting facts. These facts are those which contradict the normal believes.

The method which is presented is formalized in First Order Logic. This simple and well known formalism has a precise semantics. It was slightly extended with the introduction of a new logical operator, called pseudo-imply, which is used to represent heterogeneous transformed queries with a single formula.

A first prototype of the Cooperative Answering method has been implemented in Prolog. The straightforward implementation of logical axioms in Prolog introduces some changes in their semantics, due to the negation evaluation by failure. Indeed the theories which are considered are not complete and contain non Horn clauses. However these changes have few consequences, and don't introduce inconsistencies.

## References

- [1] J. F. Allen and D. J. Litman. Plan, goals and natural language. Research review, Computer Science and Engineering. University of Rochester, 1986.

- [2] J. F. Allen and C. R. Perrault. Analysing intention in utterance. *Artificial Intelligence*, 15(3):143–178, 1980.
- [3] K. Bowen and R. Kowalski. Amalgamating language and metalanguage in Logic Programming. In Clark, editor, *Logic Programming*, Tårnlund, 1980.
- [4] E. F. Codd. Relational Completeness of Data Base Sublanguages. In R. Rustin, editor, *Data Base Systems, Courant Computer Science Symposium 6*, pages 65–98. Prentice-Hall, 1972.
- [5] F. Cuppens and R. Demolombe. Cooperative Answering: a methodology to provide intelligent access to Databases. In *Second International Conference on Expert Database Systems*, Tysons Corner, Virginia, 1988.
- [6] R. Demolombe. Syntactical characterization of a subset of domain independent formulas. Technical report, ONERA-CERT, 1982.
- [7] R. Kowalski. The limitations of Logic. In J. Schmidt and C. Thanos, editors, *Proc. Workshop on Foundations of Knowledge Base Management*, Crete, 1986.
- [8] R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. Springer Verlag, 1983.
- [9] L. Siklossy. Impertinent question-answering: justifications and theory. In *Proc. ACM National Conf.*, pages 39–44, 1978.

## A List of Meta Predicates used in this paper

To help the reader, we give below the list of the meta predicates which are used in this paper.

- *Entity-type* ( $x$ ), *Attribute-value-type* ( $x$ ): These predicates introduce two kinds of type: the Entity type and the attribute value type.
- *Type* ( $x, y$ ): For the predicate  $x$ ,  $y$  is the list of argument types.
- *Att* ( $x, y$ ):  $x$  is an attribute of  $y$  where  $y$  is an entity type or a relationship.
- *ISA* ( $x, y$ ):  $x$  is a sub-case of  $y$ .
- *DIS* ( $x, y$ ):  $x$  and  $y$  have no common element.
- *Topic* ( $x$ ):  $x$  is a “topic”.
- *Att-Top* ( $x, y$ ):  $x$  is an attribute or a relationship related to the topic  $y$ .
- *Query* ( $x$ ), *Entity* ( $x, y$ ), *Condition* ( $x, y$ ), *Ret-Att* ( $x, y$ ): Predicates introduced to represent a query.
- *Appear* ( $x, y$ ): The attribute or the relationship  $y$  appears in the query  $x$ .
- *Int-Top* ( $q, top$ ): The topic  $top$  is interesting for the query  $x$ .
- *Normal-Belief* (“ $E(x)$ ”, “ $a$ ”, “ $C(x)$ ”): For any entity  $x$  of type  $E$ , all users assume that the condition  $C(x)$  holds; if it is not the case, the value of the attribute  $a$  for  $x$  has to be provided.
- *Rel-Att* ( $x, y, z$ ): It is relevant to provide the value of the attribute  $z$  for the entity  $y$  when it appears in the Entity part of a query  $x$ .
- *atom-entity* ( $x, y$ ): The atomic formula  $y$  appears in the Entity part of the query  $x$ .
- *entity* ( $x, y$ ): The entity type  $y$  is not disjointed with some entity type which appears in the Entity part of the query  $x$ .
- *Rel-Att-for-all* ( $x, y$ ): The attribute  $y$  is relevant for all the entity types which appear in the Entity part of the query  $x$ .
- *Rel-Att-for-some* ( $x, y, z$ ): The attribute  $z$  is relevant at least for the entity type  $y$ , which is not disjointed with some entity type which appears in the query  $x$ .
- *Add-Att* ( $x, y$ ): The formula  $y$  provides interesting additional information for the query  $y$ .

## B Completeness and Soundness proof

**Completeness:** let's consider a set of axioms *Obj-ax* at the object level of the form:  $(T(x) \longrightarrow T'(x))$ , or  $\neg(\exists(x) T(x) \wedge T'(x))$ , and the set of corresponding axioms *Meta-ax*, at the meta level, of the form: *ISA*( $T, T'$ ) or *DIS*( $T, T'$ ). Moreover, let  $F$  be a set of facts *Entity-type*( $T_1$ ) . . . *Entity-type*( $T_n$ ) where  $T_1 \dots T_n$  is the set of predicates which appear in *Obj-ax*. Then any theorem, at the object level, of the same form, which is derivable from *Obj-ax*, has a corresponding theorem at the meta level which is derivable from *Meta-ax*,  $S$  and  $F$ .

**Proof:** Let's assume  $Obj-ax \vdash \forall x(T_1(x) \longrightarrow T_2(x))$ .

Then, there are two possibilities.

1. The clause  $\neg T_1(x) \vee T_2(x)$  is a tautology. This implies that  $T_1 = T_2$ .

In that case, there is a corresponding derivation in  $\{Meta-ax, S, F\}$ . Indeed, we have *Entity-type*( $T_1$ ) and from  $R_1$  we get *ISA*( $T_1, T_1$ )  $\square$ .

2. The clause  $\neg T_1(x) \vee T_2(x)$  is derivable using Resolution principle from *Obj-ax*.

There are two cases where the resolution inference rule can be applied to two clauses in *Obj-ax*.

The first case is when it is applied to two clauses like:

$$\neg T(x) \vee T'(x) \text{ and } \neg T'(x) \vee T''(x)$$

The resolvent is :  $\neg T(x) \vee T''(x)$ , which has the same form as the clauses in *Obj-ax*.

The second case is when it is applied to two clauses like:

$$\neg T(x) \vee T'(x) \text{ and } \neg T'(x) \vee \neg T''(x)$$

The resolvent is:  $\neg T(x) \vee \neg T''(x)$ , which has the same form as the clauses in *Obj-ax*.

For each inference step in this theory, there is a corresponding inference step in  $\{Meta-ax, S, F\}$ .

Indeed, in the first case we have:

$$ISA(T, T') \text{ and } ISA(T', T'')$$

and from R2 we get: *ISA*( $T, T''$ ).

In the second case, we have:

$$ISA(T, T') \text{ and } DIS(T', T'')$$

and from R4 we get: *DIS*( $T, T''$ ).

Therefore, there is a proof of *ISA*( $T_1, T_2$ ) from  $\{Meta-ax, S, F\}$  which corresponds to the proof of  $\neg T_1(x) \vee T_2(x)$  from *Obj-ax*  $\square$ .

The same reasoning applies to theorems of the form  $\neg T_1(x) \vee \neg T_2(x)$   $\square$ .

**Soundness:** each theorem of the form *ISA*( $T, T'$ ) or *DIS*( $T, T'$ ) derivable from *Meta-ax*,  $S$  and  $F$  at the meta level, has a corresponding theorem, at the object level, of the form:  $(T(x) \longrightarrow T'(x))$  or  $\neg(\exists(x) T(x) \wedge T'(x))$ , derivable from *Obj-ax*.

**Proof:** Let's assume we have:  $Meta-ax, S, F \vdash ISA(T_1, T_2)$ .

*Meta-ax* and  $F$  are two sets of ground formulas formed with the predicates : *Entity-type*( $x$ ), *ISA*( $x, y$ ) or *DIS*( $x, y$ ).

All the facts relative to  $ISA(x, y)$  derivable from  $Meta-ax$ ,  $S$  and  $F$  can be obtained by applying a standard forward strategy.

Then, it is easy to check that each application of a rule in  $S$  has a corresponding derivation in  $Obj-ax$   $\square$ .

The same reasoning applies to theorems of the form  $DIS(T_1, T_2)$   $\square$ .

## C Example of program execution

The query presented in this annex refers to the example which is used along the paper. It shows an initial query, and a transformed query which provides the value of additional attributes. The answers have the form of formulas, instead of tuples. We use the syntax of predicate calculus where “&” denotes the conjunction, “ $\Rightarrow$ ” denotes the pseudo-imply  $\Rightarrow$  which has been introduced in previous sections and “Not” denotes the negation.