

Cooperative Answering: a methodology to provide intelligent access to Databases¹

F.Cuppens R.Demolombe

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

Abstract

The intent of this paper is to present a method to answer question in the same way as a person do, and not as a machine do. This method, called “Cooperative Answering”, is based on the idea that to provide interesting answers to a query, it is very useful to recognize what are the user’s intentions in asking such a query. The methodology, and its formalization, are presented in an informal way, using examples. However a precise formalization has been developed in First Order Logic, by distinguishing Object level, and Meta level. In this paper we consider only the semantic aspect of the information, we are not interested on linguistic aspects, such as natural language representation.

1 Introduction

In the context of traditional applications devoted to the management of a company, like payroll computation, people or programs that have to access data in a Database have a very precise definition of the data they want to access. This definition is, for example, a Relational query, if they use a Relational DBMS.

There are many other applications where people want to access data in order to take a decision, or to solve a problem whose solution cannot be found applying a simple algorithm. Systems like Decision Support Systems (DSS), or Advice Giving Systems (AGS), have been designed in order to help users in this context. An important feature of this context, from the point of view of data retrieval, is that users do not have a precise idea of the data which can help them to solve a problem, or to make a decision.

The objective of this paper is to present a methodology, along with its implementation, which is more relevant in this context than a standard access to a Relational DBMS. This methodology tries to simulate the behaviour of a person who wants to help as much as possible an interlocutor who asks them a question. That is, to try to understand why this interlocutor asks a particular question, and to determine what interesting information, not explicitly requested, he could provide him, in addition to the answer proper. This methodology is called Cooperative Answering.

We present examples in the following to illustrate this methodology. Let us consider a person who wants to organize a trip from Paris to New York and asks the question:

¹This work was supported by the ESPRIT project ESTEAM-316

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 has a long experience in giving advice to his clients, could be, for example:

Flight	Departure-time	Departure-city	Arrival-city
AF001	10h00	Paris	New York
AF001	11h00	Paris	New York
AF015	10h30	Paris	New York
AF020	10h30	Paris	Washington
SN017	10h00	Brussels	New York

This answer is not the exact answer, since it contains flights whose departure-city is not Paris, or whose arrival-city is not New York. However an expert in travel knows that, even if the conditions in the question are not exactly satisfied by the flights AF020 and BA017, these flights satisfy a neighbourhood condition, such that the change in the departure-city or arrival-city is small with respect to the length of the trip. Moreover this expert knows that a client can be interested to have a larger set of possibilities to select his flight, in particular because sometimes it is globally less expensive to start from another city, or to arrive in another city, because there can be large price differences between the airlines.

It must be noticed that since the conditions in the question have been modified, it is necessary to provide additional information telling the client which condition(s) is (are) satisfied by each flight. That is why it is necessary to indicate, in addition, the departure-city and the arrival-city.

Another expert answer could 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	

In this answer the arrival-time is an additional piece of information. This information can be interesting for a person who is organizing a trip, for example, if he has to make a connection. The validity period indicates why the flight AF001 has two different departure times. In the case where a flight is not available every day of the week it is useful to show which days it is. Finally if a flight is very expensive it is interesting to tell the client the price, even if he is not usually motivated by the price (he might be travelling for business purposes). In our example the aircraft corresponding to the flight AF001 is the Concorde, and its price is roughly three times the usual price, and it is very useful to inform the client of that.

Let us consider now the other question:

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

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

Flight	Departure-time
AF638	08h05
AF642	09h25

The answer provided by an expert might be:

Flight	Departure-time	Arrival-time	Price	Works-on
AF638	08h05	08h55	1850FF	1,2,3,4,5
AF642	09h25	10h25	1850FF	

The answer could also be:

Train	Departure-time	Arrival-time	Price
SNCF281	07h48	10h48	450FF
SNCF483	10h08	13h48	450FF

The reason why the expert might suggest taking a train is that the distance from Paris to Brussels is such that $Distance(Paris, Brussels) < 400km$ so there is no important difference, with respect to the duration of the trip, between trains and flights. On the other hand, the price difference is great and the expert, to justify suggesting a train, additionally provides the price of trains and flights.

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

These transformations can be presented more easily using the concepts of Entity and Attribute of entity, as in the Entity-Relationship Model [6]. In general the answer to a query provides some attribute values of entities satisfying some conditions. A query can therefore be decomposed into three parts:

- The type of entities which are expected in the answer. This part is called the “Entity”.
- The condition which entities of this type must satisfy. This part is called the “Condition”.
- The attribute of these entities whose values are expected in the answer. This part is called “Retrieved Attributes”.

Using this query structure the query transformations fall into the three following categories:

- Entity transformation: for example when the Entity type Flight is replaced by the Entity type Train.
- Condition transformation: for example when the arrival-city, or the departure-city are changed.
- Retrieved Attribute transformation: for example when the arrival-time, or the arrival-city are added to the list of attribute to be retrieved.

Then the additional information provided in the answers falls into the corresponding categories:

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

Roughly speaking, if answers are considered as tableaux, the extensions to the exact answer can be viewed as an extension of the tableaux in width (additional attribute values) or in height (neighbourhood conditions), or as an extension of the type of the entities which are in the tableaux.

One can also remark that an extended answer may be represented by several tableaux having different structures.

To define the methodology which simulates an expert’s behaviour we must first understand why he chooses to provide certain additional information to the client. That will be presented in the next section. In the third section we present the concepts and the methodology used to implement this simulation. The fourth section contains a refinement of the third in the way of formalization. And the fifth section shows, using examples, how an inference mechanism can derive the transformed query providing additional relevant information.

2 Why provide additional information?

It is important in defining an efficient cooperative answering method, to understand the general reasons why an expert decides to provide such additional information.

The basic idea is that when a person asks a question he is not interested in knowing the answer just to increase his knowledge, but he has the intention of performing some action, and that the answer contains information necessary or useful for realizing this action. In our example we can assume that the client asks questions about the flights not just for the pleasure of knowing the time tables of the flights, but because he has the intention of taking a plane.

If we accept the basic idea that there is always an underlying intention behind each question, then the expert who wants to be cooperative must try to recognize this intention, in order to determine the most appropriate reaction implicitly expected by interlocutor.

The problem of how to recognize user’s intentions is extensively addressed by researchers in dialogue management, or more generally in natural language understanding [13]. Though we are neither concerned here with natural language understanding, nor with dialogue management per se, we can partially reuse the methodology when queries and answers are represented in formal languages.

One method [1, 2] of recognizing user’s intentions is to assume that an expert knows a set of predefined sequences of actions, called plans, the clients may want to realize. Then, when a client asks a question, the expert tries to attach this question to an action. If he succeeds, he then assumes that the user’s intention is to perform this action, and the other actions which belong to the same plan. For example, if the client asks questions about flights, it is assumed that he wants to take a plane, and another action belonging to the same plan is, for example, to buy a ticket.

According to this point of view, the appropriate reaction of the expert is to provide the additional information which can be useful in performing this action or others in the same plan. That is why it is useful to provide the price of the ticket, even if it is not explicitly requested.

Generally, the answer to a question can be useful not only in executing a plan, but also to help in building or modifying a plan. For example, if the person who asks the first question wants to go to eastern North America, the extended answer offers other possibilities than starting from Paris and arriving in New York. In the second example the person can change his plan and take a train instead of a plane.

3 How do we simulate an expert who provides additional information?

If we want to design a system having similar behaviour that of the expert we have to characterize the kind of knowledge the expert uses, and his type of reasoning. In the following we present notions which seem to be useful for representing this knowledge.

To represent the Data Base contents it is helpful to introduce the Entity-Relationship model concepts [6]: **Entity type**, **Attribute of entity**, **Entity relationship**. In addition the concepts of **aggregation** and **generalization**, introduced in the Semantic Data Model [4, 5] are useful for a global representation of the objects a user is interested in. For example, from the Entity types Flight and Hotel, we could define the Entity type Accomodation. The Entity type Accomodation may be used to infer that, when a user ask a question about flights, it could be interesting for him to have information about the hotels which are located in the town where he wants to go. The Entity types can be structured with the **ISA relation** to support similar reasoning than for aggregation. For instance the Entity types Flight and Train are specific cases of the Entity type Means-of-Travel, and trains may be proposed instead of flights because they are also a special case of means of travel.

Moreover to recognize the user's interests we have introduced the concept of Topic, which characterizes a semantic field. Topic is more abstract than the notion of Predicate or Relation. Several predicates may be related to the same Topic; for example, the predicates corresponding to the attributes Departure-time and Arrival-time may be related to the topic Time-table. So, when a user asks a question about the Departure-time, the expert can infer that he is interested by the topic Time-table, and provides the user with the attribute values of the other attributes related to this Topic; in this case it is the Arrival-time. Though this type of reasoning is quite crude, it provides a rough characterization of data which are possibly interesting. In the area of Data Retrieval, topics are sometime called "key-words". The Topics are structured with the ISA relation, like the Entity types.

To be able to provide the additional information interesting to a given type of user, it is useful to have a **user's representation**. This representation, even if quite trivial, like a set of user types, allows a more precise definition of interesting information. For example, if we have the user types Tourist and Businessman, we can have rules expressing that the ticket price is interesting information for a Tourist, while the topic Time-table is an interesting topic for a Businessman.

A large part of the expert's knowledge is represented by rules. There are two kinds of rules: those which are independent of a given application domain, and correspond to the expertise in Cooperative Answering, in general; and those which depend on the application domain (a travel agency, in our example).

Examples of domain independent rules are:

- *If the question concerns entities of type E , and it contains a given attribute A , and this attribute is related to the topic T , then the topic T is interesting in the context of this question*
- *If the topic T_1 is interesting for the question, and T_2 ISA particular case of the topic T_1 , then the topic T_2 is interesting for this question*

Examples of domain dependent rules are:

- *If the user is a Tourist, then Cost is an interesting topic*
- *If the question concerns flights, and the flight is very expensive, then the attribute Price is a relevant attribute for this question*

To summarize: when an expert wants to be cooperative he can use a user's representation, a set of rules (which may or may not depend on the application domain), and the query, to determine the interesting topics for a given user; then he can use a high level representation of the Data Base contents to transform the query and/or to generate queries whose answers provide interesting additional information (See figure 1).

4 Sketch of the method formalization

In the previous section, we have presented notions which seem to be useful for representing the queries and the contents of the knowledge base. We are now interested in the definition of a language and its semantics to represent these notions. We have defined this semantics in a framework which is very well defined and accepted by a lot of researchers, which is Mathematical Logic.

However, even within the Logic framework, there are many different ways of formalizing the knowledge content. The basic idea, in the proposed formalization, as suggested by R.Kowalski in [9], is to deal only with standard First Order Logic (FOL). However the concepts involved in FOL have a poor semantics. For example the concepts of Entity type, Attribute of entity, Entity relationship, Attribute of entity relationship, are all represented by Predicates in FOL. And, by representing these concepts with predicates, we lose an important part of their semantics. The semantics attached to these concepts appeared, in the previous section, when we defined how different are considered, for example, an Entity type and an Attribute, in the query representation, or in the rules.

Our approach is to reflect this semantics in a Meta-Theory, also represented in FOL. In this Meta-Theory are expressed the facts and rules defining the semantics of the predicates. We thus distinguish between two language levels:

- The Object level, in which are 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.

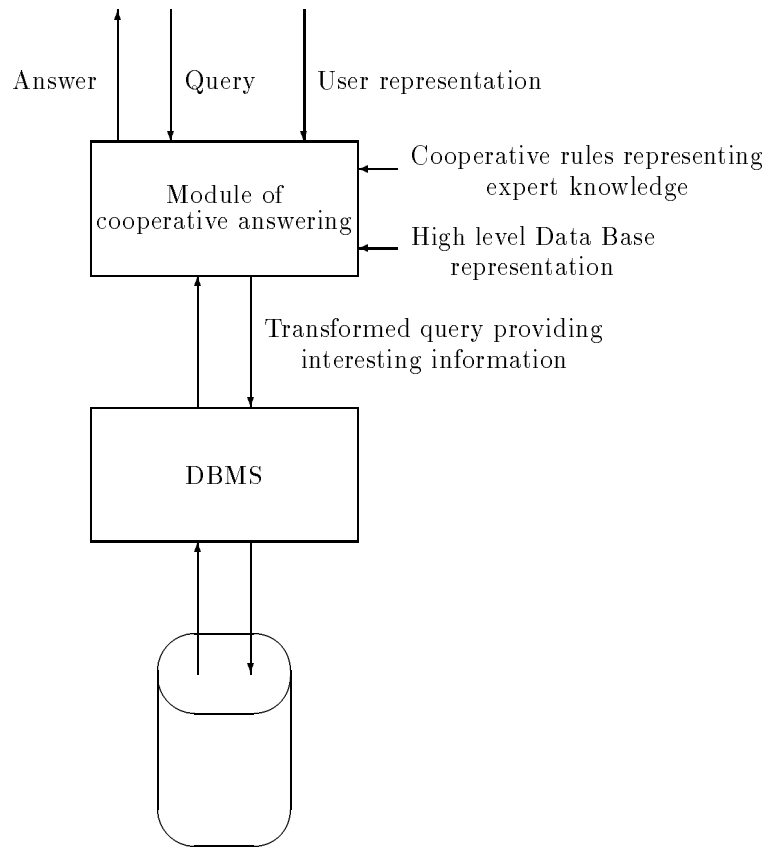


Figure 1: System organization

In the next subsections, we begin by presenting the Object language and Meta language used to represent the knowledge base information. Then, we describe our query language and the form of the answers to these queries. Thereafter the information concerning user's representation is briefly introduced, and we end this section by a presentation of the expert's knowledge used to provide additional information.

4.1 Knowledge Base representation language

4.1.1 Object language and Meta language

As said before the Object language is a First Order Language whose predicates are used to represent the facts stored in the Data Base. For instance the predicates:

$$\begin{aligned}
 & \textit{Departure-city}(x,y), \textit{Arrival-city}(x,y), \textit{Departure-time}(x,y) \\
 & \textit{Arrival-time}(x,y), \textit{Price}(x,y)
 \end{aligned}$$

represent corresponding attributes, and the predicates:

Means-of-Travel(x), Flight(x), Train(x)

represent entity types. The facts are represented by atomic formulas such as:

Departure-time(AF001, 10h00), Departure-city(AF001, Paris)

and questions are represented by formulas like:

*Flight(x) ∧
Departure-city(x,Paris) ∧ Arrival-city(x,New York) ∧
Departure-time(x,y)*

At the Meta level, facts or queries of the Object level are considered concrete objects represented by constants or terms. The correspondance between the two levels is defined by a coding function wich assigns a code to each formula of the Object level [3]. For instance the Meta level functions *pred*, *var* and *const* respectively assign a code to predicate symbols, variable symbols or constant symbols of the Object level. The functions *atom* and *and* assign a code to an atomic formula, or to the conjunction of two formulas.

Using these coding functions the Object formulas:

*Departure-city(x,Paris)
Arrival-city(x,New York)*

are represented by the terms:

*t1: atom(pred(Departure-city),var(x),const(Paris))
t2: atom(pred(Arrival-city),var(x),const(New York))*

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

and(t1,t2)

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

“Departure-city(x,Paris) ∧ Arrival-city(x,New York)”

Such terms can contain Meta variables to represent an Object formula which is not completely specified. The predicates of the Meta language are presented in the following subsections; the Meta language is a First Order language built with these predicates. The query transformation is defined by derivations at the Meta level, and there is no need in these derivations to use Object level knowledge.

4.1.2 Representation of the Entity-Relationship Model

At the Object level all the predicates have the same status. However at the Meta level they are distinguished with the Meta predicates Entity-type and Att. For example:

- *Entity-type(Flight)*

- *Entity-type(Train)*
- *Entity-type(Means-of-Travel)*
- *Att(Price,Train)*
- *Att(Departure-city,Means-of-Travel)*
- *Att(Arrival-city,Means-of-Travel)*

are facts of the Meta theory, expressing, for example that the predicate *Train* defines an Entity type, and that the predicate *Price* defines an attribute of the entities *Trains*. The quotes between Object predicates symbols are omitted when there is no risk of misunderstanding.

The Entity type structure is represented with the Meta predicate *ISA*. For example:

- *ISA(Flight, Means-of-Travel)*
- *ISA(Train, Means-of-Travel)*

express that *Flight* and *Train* are special cases of *Means-of-Travel*.

We have the following rules:

$$R1: \forall(x) \text{ Entity-Type}(x) \longrightarrow \text{ISA}(x,x)$$

$$R2: \forall(x,y,z) \text{ ISA}(x,y) \wedge \text{ISA}(y,z) \longrightarrow \text{ISA}(x,z)$$

meaning that *ISA* is a reflexive and transitive relation.

We have also:

$$R3: \forall(a,t,t') \text{ Att}(a,t) \wedge \text{ISA}(t',t) \longrightarrow \text{Att}(a,t')$$

expressing inheritance of attribute from an Entity type to a sub-type.

4.1.3 Topic notion and attributes clustering

We have also introduced the notion of “topic”.

Topics are associated to attributes and relationships and allow representation of some semantic links between attributes. In order to represent this notion we introduce a predicate *Att-Top* in the meta-language which relates a topic and an attribute of the object language.

So, if we take again the “travel-agency” example shown in the introduction, we can introduce the following topics:

Localization, Time-table, Cost, Validity-condition, Time

These topics are defined as meta-language constants. To assert that these constants are topics, we introduce the following facts into the meta-theory:

- *Topic (Cost)*
- *Topic (Time)*
- *etc.*

We can associate some topic with attributes defined for Flight by introducing the following facts into the meta-theory:

- *Att-Top (Departure-city, Localization)*
- *Att-Top (Arrival-city, Localization)*
- *Att-Top (Departure-time, Time-table)*
- *Att-Top (Arrival-time, Time-table)*
- *Att-Top (Price, Cost)*
- *Att-Top (Tariffing, Cost)*
- *Att-Top (Works-on, Validity-condition)*
- *Att-Top (Validity, Validity-condition)*

where *Departure-city*, *Arrival-city*, *etc.* exhibit actually codes in meta-theory of object predicates *Departure-city*, *Arrival-city*, *etc.*

Of course, topics can be structured in the same way as entities. For example, we could introduce in meta-theory the following facts:

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

where *ISA* is the same relation as the one used to structure entities.

4.2 Query language

4.2.1 Structure of the queries

The form of the queries is strongly restricted, as defined in the previous section. The general form is:

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

where:

“*Entity*” is a formula with only the logical operators \wedge (conjunction) or \vee (disjunction), and where predicates are only monadic predicates corresponding to Entity types.

Let $A = Var(Entity)$ (set of the free variables which appear in the “*Entity*” formula).

“*Condition*” is any formula of the object language.

Let $B = Var(Condition)$.

“*Retrieved Attributes*” is a formula with only \wedge logical operator and where the predicates are only predicates corresponding to Attributes.

Let $C = Var(Retrieved\ Attributes)$.

- We introduce the following additional condition:
 $B \subset (A \cup C)$ (The free variables of “*Condition*” must be free variables of “*Entity*” or “*Retrieved Attributes*”)

To represent and to manipulate a query, we have introduced the following predicates into the meta-language:

Query (x): x is a query.

Entity (x,y): y is the code of a formula which represents the entity part of a query x .

Condition (x,y): y is the code of a formula which represents the condition part of a query x .

Ret-Att (x,y): y is the code of a formula which represents the retrieved attributes part of a query x .

For example, to represent the query:

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

we introduce the following facts into the meta-theory:

Query ($qt1$)

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

($qt1$ is the code of the conjunction of the formulas which define the Entity, Condition and Retrieved Attributes parts of the query)

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)")

4.2.2 Structure of the transformed query

The general form of the transformed query is the same as the initial query except that we can use an extra logical operator, denoted \implies , in its "Retrieved Attributes" part.

First, we present the motivation for this new logical operator.

Let us consider for example the simple query $qt1$:

$qt1$:

Entity: Flight(x)

and let us also assume that, applying an expert rule, we must generate a new query which provides the price of flights only for those which are expensive. Let *Expensive*(x) be a formula saying that a flight is expensive. Then the transformation result could be two queries $qt2$ and $qt3$:

qt2:

Entity: $Flight(x)$

Condition: $\neg Expensive(x)$

qt3:

Entity: $Flight(x)$

Condition: $Expensive(x)$

Retrieved Attributes: $Price(x,y)$

which correspond, in FOL notation, to:

qt2: $Flight(x) \wedge \neg Expensive(x)$

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

It will be noticed that these two queries cannot be represented by a single query, like:

$$(Flight(x) \wedge \neg Expensive(x)) \vee (Flight(x) \wedge Expensive(x) \wedge Price(x,y))$$

which is equivalent to:

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

Indeed the answer to this query is not the union of the two above queries, because the two operands of the “ \vee ” operator does not have the same free variables. Moreover, the above formula has no intuitive meaning because it is not a Domain Independent formula [7].

On the other hand it would be quite desirable to have only one answer presented by a unique tableau, even if this answer is heterogeneous, in the sense that all the tuples do not have the same component number. Here some tuples are $\langle x \rangle$ values, others are $\langle x,y \rangle$ values. And the tuples structure depends on the evaluation of $Expensive(x)$.

The extra logical operator \implies is introduced to be able to represent a set of heterogeneous queries, that is, queries which do not have the same free variables, with a unique query.

The semantic of this operator is the following:

If $(f \implies g)$ is a sub-formula of a formula F, then, for a given instantiation of the free variables in F, $(f \implies g)$ is true iff $(f \wedge g)$ is true or f is false.

Moreover if f is true, the variables which are considered to be free in the definition of the tuple structure in the answer are either the free variables of f and g or the free variables of f .

This last comment differentiates between the usual implication operator \longrightarrow and the operator \implies .

In the following, the queries containing the \implies operator are called “heterogeneous queries”: Using this operator the above queries qt2 and qt3 are represented by this following query:

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

4.2.3 Structure of the answers

In this part we define the general form of an answer to a query.

The answer to a query is usually defined in the following way [12]:

- Let L be a first-order language.
- Let $F(X)$ be a well-formed-formula of L . The only free variables of F are $X = \langle x_1, \dots, x_n \rangle$.
- Let T be some theory of L .

The answer to the query $F(X)$ in the theory T is the set of tuples of constants $C = \langle c_1, \dots, c_n \rangle$ such that $F(\sigma(C/X))$ is a theorem of T :

$$Answer = \{C | T \vdash_L F(\sigma(C/X))\}$$

where $\sigma(C/X)$ means that C is substituted for X .

Concerning our query language, such a definition of an answer to some question does not provide enough information to the user, as we can show in the following example:

Let us consider a Data Base in which is represented an entity class *Train*.

We consider two sub-classes of *Train*:

Train-TEE: Set of Trans Europ Express Trains.

Train-Sleeping: Set of trains having sleeping cars.

The following attribute is associated with the class of *Train-TEE* entities:

Surcharge-TEE: For the TEE train x , there is a surcharge y .

The following attribute is associated with the class of *Train-Sleeping* entities:

Surcharge-sleeping (x, y): For the train with sleeping cars x , there is a surcharge y .

Let us consider the following heterogeneous query:

Entity: *Train* (x)

Retrieved attributes:

$$\begin{aligned} & (Train-TEE(x) \implies Surcharge-TEE(x, y)) \wedge \\ & (Train-sleeping(x) \implies Surcharge-sleeping(x, z)) \end{aligned}$$

which corresponds in natural language to the following query:

“For all Trains, retrieve Surcharge-TEE for Train-TEE, and Surcharge-sleeping for Train-sleeping”

In tuple structure, the answer to this question is constituted by the following sets:

- A set S_1 of singletons $\langle x \rangle$ corresponding to Trains which are neither Train-TEE nor Train-sleeping.
- A set S_2 of pairs $\langle x, y \rangle$ corresponding Train-TEE.

- A set S_3 of pairs $\langle x, z \rangle$ corresponding to Train-sleeping.
- A set S_4 of triplets $\langle x, y, z \rangle$ corresponding to Train-TEE with sleeping cars.

With this kind of answer, the user cannot know if some pair $\langle a, b \rangle$ belongs to S_2 or S_3 . This is one reason for saying that providing answers in tuple structure does not provide enough information to the user.

In our system, the answer corresponding to Train-TEE would be represented by the following formula:

$$\text{Train-TEE}(a) \wedge \text{Surcharge-TEE}(a, b)$$

while answers corresponding to Train-sleeping would be represented by the following instantiated formula:

$$\text{Train-sleeping}(a) \wedge \text{Surcharge-sleeping}(a, c)$$

The user can thus easily distinguish distinct solutions.

In the general case, formulas which are provided in the answer are conjunctions of ground formulas (positive or negative). We have already adapted the definition of satisfiability of formulas, and we are defining an algebra on this set of formulas, instead of on a standard set of tuples.

4.3 User representation

We have already observed in preceding sections that to have cooperative behaviour, the system must know some facts about the user. In interactions between two humans, each obtains information about the other during the dialogue. In the case of interactions with a computer system, this can be simulated by a dialogue manager which can ask the user questions. In this paper we ignore the dialogue management aspects, and we assume that the user representation has been previously determined.

A first user representation subset is expressed by elementary facts (which could be formalized in propositional logic). For example, we can easily formalize the following informations:

- User travels for tourism reasons.
- User is a businessman.
- User is in a hurry.
- *etc.*

For this, it is sufficient to define a user-type set. So, we could have the following types:

- *User-type (Tourist)*
- *User-type (Businessman)*
- *User-type (Man-in-a-hurry)*

It will be also important to be able to use information concerning some user attributes such as:

- User is French.

- User is 50 years old.

This kind of information can be easily formalized in first-order logic by defining some user classes, with Nationality and Age attributes. For this, it is sufficient to introduce the following predicates into the language:

User (x): x is a user.

Nationality (x,y): x has y for nationality.

Age (x,y): x is y years old.

Another information set is composed of user knowledge and belief. For example:

- User believes that airplane travel is dangerous.
- User knows that Concorde crosses Atlantic ocean in two hours and a half, and that it is an expensive flight.

This information can also be used by the cooperative answering module to provide user with additional information. Indeed, let us suppose that a user knows that Concorde is an expensive and fast flight. In these conditions, if Concorde is a query solution, it is sufficient to specify it to user. Then it is useless to tell him that it is an expensive flight, since he can infer this information.

It can be noticed at once that information about knowledge and belief cannot usually be formalized in first order logic. A classical way to represent this information consists in introducing a modal logic (namely epistemic logic [8]).

At this time, information about user used to provide him some additional information, is that which can be formalized in first-order logic. We have not yet studied the problem of formalizing and using information about user's knowledge.

4.4 Expert knowledge representation

Expert knowledge contains general rules which define how the concepts such that: Topic, Entity, Attribute of entity,..., can be used to derive what Topics are interesting, and what additional attributes may be provided to the user. In the following we present some of these rules which are used in the next section for an example, but it would be too long to present all of them in this paper.

The rule R4 expresses that, if the attribute a appears in the query q , and is related to the Topic top , then this Topic is interesting for the query q .

$$R4: \forall (q, a, top) \\ (Appear (q, a) \wedge Att-Top (a, top)) \longrightarrow Int-Top (q, top)$$

The rule R5 expresses that if the Topic top is interesting for the query q , and top' is a special case of the Topic top , then top' is also interesting for q . For example, if *Time* is interesting for q , then *Time-table* is also interesting.

$$R5: \forall (q, top, top') \\ (Int-Top (q, top) \wedge ISA (top', top)) \longrightarrow Int-Top (q, top')$$

The rule R6 expresses that, if the query q concerns the entity e , and a is an attribute of e related to an interesting Topic, then it is relevant, for the query q , to provide in addition the attribute a .

$$\begin{aligned}
R6: & \forall (q, e, v, top, a) \\
& (entity(q, "e(v)") \wedge \\
& Int-Top(q, top) \wedge \\
& Att(a, e) \wedge \\
& Att-Top(a, top)) \\
& \longrightarrow Rel-Att(q, "e(v)", "a(v, v1)")
\end{aligned}$$

The rule R7 expresses that, if f is a relevant attribute of e' for the query q , and the query q concerns entities e which are special cases of entities e' , then the attribute f can be added into the list of attributes which must be retrieved. That is, the notion of relevant-attribute is inherited in the structure of Entity types. For example if *Arrival-time* is relevant for *Means-of-Travels*, it is also relevant for *Flights*.

$$\begin{aligned}
R7: & \forall (q, e, e', v, f) \\
& entity(q, "e(v)") \wedge Rel-Att(q, "e'(v)", f) \wedge ISA(e, e') \\
& \longrightarrow Add-Att(q, f)
\end{aligned}$$

The rule R8 expresses that, if the attribute f of the entity e' is relevant for the query q , and the query concerns also other entities e , which are not special cases of e' , then the attribute f must be provided only for entities of the type e' .

For example if the Entity part of the query q is: $Flight(x) \vee Train(x)$, and the attribute f is relevant for trains, its value must be provided only for trains.

$$\begin{aligned}
R8: & \forall (q, e, e', v, f) \\
& entity(q, "e(v)") \wedge Rel-Att(q, "e'(v)", f) \wedge \neg ISA(e, e') \\
& \longrightarrow Add-Att(q, "e'(v) \implies f")
\end{aligned}$$

The rule R9 expresses that if the query q concerns entities e_1 , and e_1 and e_2 are special cases of the entity e , then it is relevant to provide the entity e_2 instead of e_1 .

$$\begin{aligned}
R9: & \forall (q, x, e, e_1, e_2) \\
& (entity(q, "e_1(x)") \wedge ISA(e_1, e) \wedge ISA(e_2, e)) \\
& \longrightarrow Rel-Entity(q, "e_1(x)", "e_2(x)")
\end{aligned}$$

All the preceding rules are independent of any application and represent expertise in general Cooperative Answering. The following rules depend on the application.

The rule RD1 expresses that, if it is relevant to provide Trains instead of Flights, that must be restricted to the cases where the distance is less than 400km. That is, in the transformed query, the Condition part is transformed in order to insert this restriction.

$$\begin{aligned}
RD1: & \forall (q, x, y_1, y_2) \\
& (Rel-Entity(q, "Flight(x)", "Train(x)") \wedge \\
& condition(q, "Dep-Arr(x, y_1, y_2)")) \\
& \longrightarrow Transf-cond(q, "Flight(x)", \\
& \quad "Train(x)", \\
& \quad "Dep-Arr(x, y_1, y_2)", \\
& \quad "Dep-Arr(x, y_1, y_2) \wedge Distance(y_1, y_2) < 400km")
\end{aligned}$$

where $Dep-Arr(x,y,z)$ is an abbreviation for the formula $Departure-time(x,y) \wedge Arrival-time(x,z)$.

The rule RD2 expresses that, if the query concerns Flights, then the days for which the flight is not working, if they are, is a relevant information.

$$\begin{aligned}
 RD2: \forall (q, x) \\
 & \text{entity}(q, \text{"Flight}(x)\text{"}) \\
 & \longrightarrow \text{Rel-Att}(q, \text{"Flight}(x)\text{"}, \\
 & \quad \text{"(Day}(y) \wedge \neg \text{Works-on}(x, y)) \\
 & \quad \implies \neg \text{Works-on}(x, y)\text{"})
 \end{aligned}$$

5 Query transformation to provide additional information

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

Let us consider the following query:

Query (qt1)

Entity (qt1, "Flight(x)")

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

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

First we present the transformation which provides the values of additional attributes:

We have:

Appear(qt1,Departure-time)

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

then, from R4, we get: *Int-Top*(qt1,Time-table)

We have:

Att(Arrival-time,Means-of-Travel)

ISA(Flight,Means-of-Travel)

then, from R3, we get: *Att*(Arrival-time,Flight)

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

then from R6, we get:

Rel-Att(qt1, "Flight(x)", "Arrival-time(x,v₁)")

From RD2 we get:

Rel-Att(qt1, "Flight(x)", "(Day(y) \wedge \neg Works-on(x,y)) \implies \neg Works-on(x,y)")

From R7 we get:

Add-Att(qt1, "Flight(x)", "Arrival-time(x,v₁)")

Add-Att(qt1, “Flight(x)”, “(Day(y) \wedge \neg Works-on(x,y)) \implies \neg Works-on(x,y)”)

When new atomic formulas are inserted into the list of attributes to be retrieved, it could happen that the same variable appears in several predicate arguments, and introduces artificial links. This is the case, for example, for the variable y in Departure-time(x,y) and Works-on(x,y). To avoid this, the Rename predicate changes the variable name y in the above formula into y_1 . Finally get a new query qt2:

Query (qt2)

Entity (qt2, “Flight (x)”)

Condition (qt2, “Departure-city (x,Paris) \wedge
Arrival-City (x,Brussels) \wedge
Departure-time (x,y) \wedge (8 < y) \wedge (y < 12)”)

Ret-Att (qt2, “Departure-time (x,y) \wedge
Arrival-time (x, v_1) \wedge
(Day (y_1) \wedge \neg Works-on (x, y_1))
 \implies \neg Works-on (x, y_1))”)

We present now the transformation which provides other entities, namely the trains, in the answers.

We have:

Entity (qt1, “Flight(x)”)

ISA (Flight, Means-of-Travel)

ISA (Train, Means-of-Travel)

then from R9, we have: *Rel-Entity* (q, “Flight (x)”, “Train (x)”)

From:

Condition (qt1, “Departure-city (x,Paris) \wedge
Arrival-city (x,Brussels) \wedge
Departure-time (x,y) \wedge (8 < y) \wedge (y < 12)”)

we can infer that:

condition (qt1, “Departure-city (x,Paris) \wedge
Arrival-city (x,Brussels)”)

then, from RD1, we get:

Transf-cond (qt1, “Flight (x)”,
“Train (x)”,
“Dep-Arr (x, y_1 , y_2)”,
“Dep-Arr (x, y_1 , y_2) \wedge Distance (y_1 , y_2) < 400km”)

and we finally get a new query qt3:

Query (qt3)

Entity (qt3, “Train (x)”)

*Condition (qt3, "Departure-city (x,Paris) \wedge
 Arrival-City (x,Brussels) \wedge
 Distance (Paris,Brussels) $<$ 400km \wedge
 Departure-time (x,y) \wedge (8 $<$ y) \wedge (y $<$ 12)")*

Ret-Att (qt3, "Departure-time (x,y)")

So, after application of the transformation which provides the value of additional attributes, we transform the Retrieved Attributes part of qt3 to provide the Departure Time of the trains and we obtain the following query:

Query (qt4)

Entity (qt4, "Train (x)")

*Condition (qt4, "Departure-city (x,Paris) \wedge
 Arrival-City (x,Brussels) \wedge
 Distance (Paris,Brussels) $<$ 400km \wedge
 Departure-time (x,y) \wedge (8 $<$ y) \wedge (y $<$ 12)")*

Ret-Att (qt4, "Departure-time (x,y) \wedge Arrival-time (x,v₁ ")

6 Conclusion

We have presented a method to support intelligent access to Data Bases which is based on the idea of Cooperation between the system and the user. This method allows to provide additional relevant information. Other work, with similar objectives, is presented in the literature, but our approach is original in several respects. If we compare with J.F. Allen's work, presented in [2, 1], the main difference is that we do not use the concept of plan to recognize user's intentions, but a more abstract concept, namely the concept of Topic of interest. It does not allow the representation of such detailed intentions as those which can be represented with plans, but it allows reasoning on the user's interests even if the user has no precise plan in mind, and asks questions in order to build his plan. The main difference with A. Motro's work [10, 11] is that additional information corresponds only to the transformation of the Condition part of the question, and there is no reasoning on user's intentions. Moreover additional information is provided only in the case where the initial answer is empty.

Of course there are many open issues, one of the most important of which is how to manage transitivity. That is, the additional information can be considered as neighbour of the strict answer, and the question is: do we have to provide neighbour information of neighbour information, ..etc..? For instance, if we consider the Entity type structure, we propose alternative entities which have the same "father" as the initial one, but we could also propose entities having the same 'granfather", and so on. The same question arises about the Topic structure; if we recognize some Topic to be interesting, another Topic having the same "father", or "granfather", could also be considered as an interesting one. The question also arises for the Condition part transformation. Another issue to be investigated is the user representation, which is currently very simple, and could be

much more sophisticated. Indeed the more information the system has about the user, the more cooperation can be efficient.

Finally we point out that the method has been implemented in a prototype written in Prolog (see the example in Annex). It runs on a toy example, and the feedback from a realistic application will be very useful in making changes to the rules, if needed. However the quality of the answers strongly depends on the rules introduced to represent a specific application.

Acknowledgements

We would like to thank Tom Grossi for correcting our initial text in “French-English”.

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*, Tarnuld, 1980.
- [4] M. Brodie. On modelling behavioural semantics of data bases. In *Proc. VLDB*, 1981.
- [5] M. Brodie. On the development of data models. In *On conceptual modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer Verlag, 1983.
- [6] P. P. Chen. The entity / relationship model: toward a unified view of data. *ACM TODS*, 1(1), March 1976.
- [7] R. Demolombe. *Syntactical Characterization of a Subset of Domain Independent Formulas*. Technical Report, ONERA-CERT, 1982.
- [8] J. Y. Hintikka. *Knowledge and belief*. Cornell University Press, Ithaca, New-York, 1963.
- [9] R. Kowalski. The limitations of logic. In J. Schmidt and C. Thanos, editors, *Proc. Workshop on Foundations of Knowledge Base Management*, Crete, 1986.
- [10] A. Motro. Query generalization: a technique for handling query failure. In *Proc. First International Workshop on Expert Database Systems*, pages 314–325, Kiawah Island, South Carolina, 1984.
- [11] A. Motro. Supporting goal queries in relational databases. In *Proc. First International Conf. on Expert Database Systems*, 1986.
- [12] 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.
- [13] J. R. Searle. *Speech Acts: An essay in the philosophy of language*. Cambridge University Press, New-York, 1969.

A Annex : 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 two transformed queries, providing additional attributes, or entities of another type. The answers have the form of formulas, instead of tuples, and each atom is implicitly connected to the others by an and operator. The syntax is Lisp-like, and could be easily transformed into Predicate Calculus.