객체지향 데이터베이스 시스템에서 내포적 답의 처리 기법

김 양 희†

요 약

기존 데이터베이스 시스템에서 질의를 처리할 때, 질의문의 응답은 사실이나 튜플들의 집합이다. 객체지향 데이터베이스 시스템에서도 객체들의 집합이 질의문의 응답이다. 그러나 추론적 데이터베이스 시스템에서는 질의문에 대하여 식의 집합으로 응답할 수 있으므로, 질의문 처리 비용을 절감할 수 있으며, 데이터베이스 상태와는 독립적인 좀 더 간결한 "내포적 답"으로 표현할 수 있다. 본 논문에서는 객체지향 데이터베이스 시스템에 규칙을 도입하여 추론적 데이터베이스 시스템에서 사용되는 내포적 질의문 처리 방법과 결합하여, 기존의 객체지향 데이터베이스 시스템에서 답을 얻을 수 없었던 불완전한 질의문에 대한 답을 얻을 수 있게 하였을 뿐만 아니라, 답의 클래스의 이름으로 표현함으로써, 답을 보다 명료하게 이해할 수 있게 해준다.

Intensional Answers in Object-Oriented Database Systems

Yang Hee Kim†

ABSTRACT

When processing a query in a conventional database systems, a set of facts or tuples are usually returned as an answer. This also applies to object-oriented database where a set of objects is returned. Deductive database systems, however, provide the opportunity to obtain the answer of a query as a set of formulas, thereby reduce the costs to process the query, and represent its "intensional answers" in a more compact way independently of the database state. In this paper, by introducing rules into the object-oriented database systems and integrating the intensional query processing of deductive database systems into the object-oriented database systems, we make it possible not only to answer incomplete queries which are not able to be answered in conventional object-oriented database systems, but also to express the answer-set abstractly as the names of classes, which provides us better understanding of the answer.

키워드 : 객체지향 데이터베이스(Object-Oriented Databases), 클래스 계층(Class Hierarchy), 내포적 질의문 처리(Intensional Query Processing, 내포적 답(Intensional Answers), SLD-분석(SLD-resolution)

1 Introduction

Conventionally, a query to an object-oriented database (OODB) system is answered only by the set of objects that satisfy a given query [1]. These objects may belong to different classes within a class hierarchy or they may be different complex objects somehow related to each other.

Compared to an extensional answer, an intensional answer is a set of rules which characterizes the conditions that an object must satisfy in order to belong to an answer to a certain query. This formula representation of the output of a query is independent of particular circumstances in the database. Not only does an intensional answer represent the answer to the given query in a more compact way, but it can be computed much faster than extensional answers. Most of the time intensional answers can be computed only using the rules without accessing the database. For a detailed description of the intensional query processing, we refer to [13].

In this paper, we introduce rules into OODB systems and apply the intensional query processing (IQP) techniques to OODB systems. All the query languages in OODB systems known to us are not able to incomplete queries. An incomplete query is a query on the attribute which belongs to a subclass but not the base class. In this paper, we make it possible to answer incomplete queries by representing OODB schema in terms of rules. Conventionally, the answer-set of a query in OODB is represented as a set of objects. But, the presence of semantics in OODB schema and IQP

† 정 회 원 : Department of Physical Education, Korean National University of Physical Education (한국체육대학교 체육학과 교수)
논문접수 : 2001년 10월 8일, 심사완료 : 2002년 1월 18일

methodologies enable us to express the answer-set abstractly as names of classes. In this paper, we present an algorithm to obtain abstract representation of a given answer-set. It provides us better understanding of the answer.

Rules which represent OODB schema consists of structural rules and subclassing rules. The structural rules in turn consist of "IS_A"-relationships representing the class hierarchy. The subclassing rules represent characteristic properties of subclasses. We then transform all the rules to non-recursive Horn clauses and get an intensional answer by using SLD-resolution.

We provide some sample queries to show the advantages of an intensional answer to give query in an OODB system over conventional answers.

We organize this paper as follows. In section 2, we discuss several researches which has been done in the IQP. In section 3, we review the definition of intensional answers and methods for deriving intensional answers. In section 4, we look at the "IS_A"-relationship in OODB systems and the rules in OODB systems which are necessary in order to use the IQP. In section 5, we formalize methods to derive intensional answers for a class hierarchy model and give a detailed example. In section 6, we give conclusions and some remarks for possible extensions of the method in this paper.

2. Researches on Intensional Answers

For last several years, a lot of works have been done in this area. While each adopts different approaches, all have the common goal, that is to answer queries more abstractly. But research which integrates this area and OODB was started few years ago.

A deductive database is composed of extensional predicates (facts) and intensional predicates (rules). Cholvy and Demolombe [4] provided answers to queries that are independent of a particular set of facts, i.e., answers that are derived only from the rules. They also developed a method for generating answers using resolution.

Imielinski [6] tried to incorporate intensional predicates as a part of the answers and Pascual and Cholvy [9] restricted types of rules in intentional databases (IDB) to Horn clauses which are most widely adopted and studied in deductive database systems.

Motro [7] discussed a method that applies database constraints to generate intensioanl answers and Motro and Yuan informally discuss a simple query language incorporating intensional queries in [7].

Yoon and Song [13] [14] used only Horn-clauses for intensional databases and use a SLD-resolution which takes advantages of Horn-clauses. They introduced the notions of extended term-restricted rules, relevant literals and relevant clauses to avoid generating certain meaningless intensional answers. Also, Yoon and Song [15] introduced a partially automated method for generating intensional answers to represent answer-set abstractly for a query by extending current data mining techniques.

3. Definition of Intensional Answers

3.1 Definition

Cholvy and Demolombe [4] give a formal definition of query answers. Define T as the database theory consisting of a set of facts and rules and let $Q(X)$ be a query where X is a tuple of free variables. Then, the intensional answer $ANS(Q)$ to a certain query $Q(X)$ is defined as follows :

$$ANS(Q) = \{ans(X) : T \vdash \forall X\ (ans(X) \rightarrow Q(X))\}$$

where $ans(X)$ is a literal.

However, we want to restrict the answers within a defined domain of interest. Here are some restriction on the intensional answer set.

So, let $DP = \{P_1, \cdots, P_n\}$ be set of predicate symbols either of the IDB of the extensional database (EDB). And let $L(DP)$ be the first order language whose predicate symbols are $P_1, \cdots, P_n$. Then define an intensional answer $ANS(Q, DP)$ to the query $Q(X)$ by :

$$\begin{aligned} ANS(Q, DP) = \{&ans(X) : asn(X) \in L(DP) \text{ and} \\ &T \vdash \forall X\ (ans(X) \rightarrow Q(X)) \text{ and} \\ &(ans(X) \textit{ is not the negation of a tautology}) \text{ and} \\ &(\textit{each } ans(X) \textit{ is not redundant})\} \end{aligned}$$

3.2 Method for deriving answers

In the previous section, we defined an intensional answer. We note that :

$$\begin{aligned} & T \vdash \forall X\ (ans(X) \rightarrow Q(X)) \\ \Leftrightarrow\ & T \cup \{\text{not}(\forall X\ (ans(X) \rightarrow Q(X)))\} \textit{ is inconsistent} \\ \Leftrightarrow\ & T \cup \{\exists X(ans(X) \wedge \text{not}\,(Q(X)))\} \textit{ is inconsistent} \end{aligned}$$

Let S be a set of clauses that represent the standard clausal form of T axioms [3]. Then

$$\exists X\ (ans(X) \wedge \text{not}\,(Q(X)))$$

leads the standard form $ans(X_0) \wedge not(Q(X_0)))$ where X_0 is a tuple of Skolem constants. So, the above formula is equivalent to

$$S \cup \{ans(X_0), \neg(Q(X_0))\} \text{ is inconsistent.}$$

Therefore, answer formulas $ans(X)$ are such that resolution on $S \cup \{ans(X_0), \neg(Q(X_0))\}$ leads to the empty clause.

However, initially we do not know what $ans(X_0)$ are. So, for the resolution processing, we will start with $S \cup \{\neg Q(X_0)\}$. After resolving $S \cup \{\neg Q(X_0)\}$ will result in a resolvent $R(X_0)$ and then resolving $R(X_0)$ together with $ans(X_0)$ will result in the empty clause. That means that $ans(X_0)$ must equal to $\neg R(X_0)$.

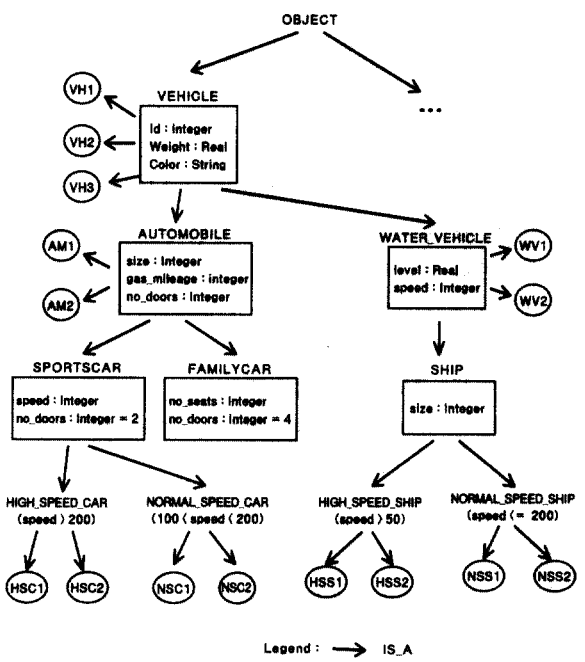
4. Class Hierarchy and Rules for abstract expression

4.1 Class Hierarchy

We look at a class hierarchy, representing "IS_A"-relationships between the different classes. We assume the following query syntax in our context :

SELECT <attribute of target record type>
FROM <object variables>
WHERE <predicate>

The class hierarchy represented in the following Figure will be our object-oriented example database.



(Figure 1) Class Hierarchy

A typical incomplete query for the class hierarchy in (Figure 1) is the following :

SELECT VEHICLE.id
WHERE speed > 50

According to query languages in OODB systems known to us, the above query is incorrect. But by integrating intensional query processing into OODB system, we can answer such a query.

In a conventional OODB we get back a set of vehicle ids where the speed of the vehicle is greater than 50 miles per hour. According to our sample database in (Figure 1), we get back the following set :

{HSC1, HSC2, ···, NSC1, NSC2, ···, HSS1, HSS2, ··· }.

All these objects belong to different subclasses of the base class object. In order to find all the applying objects, the database must provide a technique to search through all the subclasses of vehicle. But this is not the common "State of the Art" in OODB systems right now. There do not exist good query languages for OODB systems which are simple to use the advantages of an object-oriented system.

By using semantics in OODB schema and intensional query processing methodologies, the answer-set of a query is given by not a set of objects but names of classes to which answer object belong. Since these abstract representation of the answer set is more concise, it provides us better understanding of the answer set.

In our example there are the following intensional answers :

- intensional-answer1 = all high_speed_cars
- intensional-answer2 = all normal_speed_cars
- intensional-answer3 = all high_speed_ships

In the next section of this paper, we will look at a way to automatically access the desired subclasses without being aware of the exact structure of the class hierarchy.

3.2 Rules

Introducing the notion of rules we can distinguish between different kinds of rules :

- integrity constraints
- "usual" rules
- structural rules
- subclassing rules

First, an object-oriented database may have integrity constraints expressed as rules. These are not unique for

object-oriented systems, but can be found in any database system. So, we are not interested a them here.

The second sort of rules are the "usual" rules that any deductive database can contain. Also these rules can be used the same way in an object-oriented database system with rules as in deductive database systems. The rules don't depend on dealing with objects or with tuples. So the occurrence of this type of rules is also not something which is special or requires a different treatment in object-oriented database than it requires in deductive databases.

The more important rules we have to be concerned about are the rules that has to be established in order to complete an intensional query successfully within an object-oriented database system. These rules come out of the class hierarchy of the objects. The system maintains automatically a logical representation of the schema information. The rules we are interested in are the rules whose information is stored in the database schema. The rules concerning the structural information of the subclassing schema (the "IS_A"-relationship) are the subclassing rules.

The structural rules and subclassing rules of our given example database would be the followings :

- structural rules
  - "IS_A" - rules
    IS_A (automobile, vehicle)
    IS_A (watervehicle, vehicle)
    IS_A (sportscar, automobile)
    IS_A (family_car, automobile)
    IS_A (ship, watervehicle)
    IS_A (high_speed_car, sportscar)
    IS_A (normal_speed_car, sportscar)
    IS_A (high_speed_ship, ship)
    IS_A (normal_speed_ship, ship)
- subclassing rules
  - high_speed_car (X) ← sportscar (X) ∧ speed (X, Y) ∧ greater (Y, 200)
  - normal_speed_car (X) ← sportscar (X) ∧ speed (X, Y) ∧ greater (Y, 100) ∧ less (Y, 200)
  - sportscar (X) ← automobile (X) ∧ no_doors (X, Y) ∧ equal (Y, 2)
  - family_car (X) ← automobile (X) ∧ no_doors (X, Y) ∧ equal (Y, 4)
  - high_speed_ship (X) ← ship (X) ∧ speed (X, Y) ∧ greater (Y, 50)
  - normal_speed_ship (X) ← ship (X) ∧ speed (X, Y) ∧ lesseq (Y, 50)

5. Formalization of Intensional Query Processing in Object-Oriented Database Systems

In this section, we outline an algorithm deriving intensional answers for a class hierarchy model consisting of non-recursive Horn clauses. By limiting non-recursive clauses the algorithm can be terminated, and by Horn clauses efficient algorithm can be used.

5.1 Processing comparison literals

To compute intensional answers efficiently, subclassing rules should be represented in a proper form. Since testing satisfiability in first-order logic formula is undecidable, adopting first-order logic formula for managing subclassing rules is not desirable. Therefore, we need a subset of first order logic expressions which is powerful enough for expressing subclassing rules and in which the satisfiability problem can be processed efficiently.

Subclassing rules can be represented with the "simple predicates" of Eswaran et al. [7]. The BNF of simple predicate abbreviated by SP is as follows :

$$\langle SP\rangle ::= \langle SP\rangle \wedge \langle SP\rangle \mid \langle SP\rangle \vee \langle SP\rangle \mid \neg \langle SP\rangle \mid \langle predicates\rangle$$

$$\langle predicates\rangle ::= \langle comparison\ operator\rangle(\langle variable\ name\rangle, \langle constant\rangle) \mid \langle comparison\ operator\rangle(\langle variable\ name\rangle, \langle variable\ name\rangle) \mid \langle comparison\ operator\rangle(\langle variable\ name\rangle, \langle variable\ name\rangle + \langle constant\rangle)$$

$$\langle comparison\ operator\rangle ::= equal \mid not\_equal \mid greater \mid greater\_eq \mid less \mid less\_eq$$

Rosencrantz and Hunt showed that the satisfiability problem of the set of simple predicate is NP-hard [11]. But they showed that conjunctive not_equal free predicates (simple predicates that do not contain not_equal and ∨) can be solved in polynomial time. We can represent a large class of subclassing rules with conjunctive not_equal free predicates. The following algorithm changes a conjunctive not_equal free predicate to a weighted directed graph [11], [7].

Input : A conjunctive not_equal free predicate P.
Output : A weighted directed graph.
(v_1 and v_2 stand for variables and c stands for a constant)

1. Convert P into an equivalent predicate P′ containing only less_eq comparison literal as follows :
   1.1 Replace $v_1 = v_2$ with $(v_1 \le v_2 + 0) \wedge (v_2 \le v_1 + 0)$.
   1.2 Replace $v_1 < v_2$ with $v_1 \le v_2 + (-1)$.
   1.3 Replace $v_1 \le v_2$ with $v_1 \le v_2 + 0$.
   1.4 Replace $v_1 > v_2$ with $v_2 \le v_1 + (-1)$.
   1.5 Replace $v_1 \ge v_2$ with $v_2 \le v_1 + 0$.
   1.6 Replace $v_1 = c$ with $(v_1 \le 0 + c) \wedge (0 \le v_1 + (-c))$.
   1.7 Replace $v_1 < c$ with $v_1 \le 0 + (c - 1)$.
   1.8 Replace $v_1 \le c$ with $v_1 \le 0 + c$.
   1.9 Replace $v_1 > c$ with $0 \le v_1 + (-c - 1)$.
   1.10 Replace $v_1 \ge c$ with $0 \le v_1 + (-c)$.
   1.11 Replace $v_1 = v_2 + c$ with $(v_1 \le v_2 + c) \wedge (v_2 \le v_1 + (-c))$.
   1.12 Replace $v_1 < v_2 + c$ with $v_1 \le v_2 + (c-1)$.
   1.13 Replace $v_1 \le v_2 + c$ with $v_1 \le v_2 + c$.
   1.14 Replace $v_1 > v_2 + c$ with $v_2 \le v_1 + (-c - 1)$.
   1.15 Replace $v_1 \ge v_2 + c$ with $v_2 \le v_1 + (-c)$.
2. Convert P′ into a weighted directed graph. The graph has a node for each variable and a node for a constant zero. Conversion is as follows :
   2.1 $v_1 \le v_2 + c$ corresponds to an edge from node v_1 to node v_2 with edge weight c.
   2.2 $v_1 \le 0 + c$ corresponds to an edge from node v_1 to zero node with edge weight c.
   2.3 $0 \le v_1 + c$ corresponds to an edge from zero node to node v_1 with edge weight $-c$.

(Algorithm 1)

There are two restrictions in the above algorithm. The one is that each variable should be integer valued. The other is that predicates can not have not_equal operators. Fortunately, many of subclassing rules involve integer valued domains such as engine size, price, number of doors, etc. And in this paper, we will deal with not_equal free predicates.

The next algorithm will change a weighted directed graph G with no multiple edges to a conjunctive less_eq predicate (not_equal free predicate that contains only less_eq).

Input : A weighted directed graph G with no multiple edges.
Output : A conjunctive less_eq predicate.

(v_1 and v_2 stand for variables and c stands for a constant)

1. An edge from node v_1 to node v_2 with edge weight c corresponds to $v_1 \le v_2 + c$.
2. An edge from node v_1 to zero node with edge weight c corresponds to $v_1 \le 0 + c$.
3. An edge from zero node to node v_1 with edge weight $-c$ corresponds to $0 \le v_1 + c$.

(Algorithm 2)

The next algorithm will test comparison literals using a weighted directed graph and return a truth constant or a simplified predicate.

Input : A predicate consisting of the conjunction of an old comparison predicate (predicate in resolvent before resolution) and a new comparison predicate (predicate in resolvent after resolution).

Output : Truth constant (TRUE of FALSE) or simplified predicate.

1. Apply algorithm 1 to the conjunction of old and new comparison predicate. And then we get a weighted directed graph G.
2. If G has a negative cycle then
   return FALSE.
   else
   If there is more than one edge from node v_1 to node v_2 then
   begin
   retain the minimum weight edge and discard the others
   apply algorithm 2 to G and we get a conjunctive less_eq predicate using step 1 of algorithm 1
   if P is the same as new comparison predicate then
   return TRUE
   else
   return P
   end
   else
   return old comparison predicate $\wedge$ new comparison predicate

(Algorithm 3)

We can use Floyd's all shortest path algorithm to see if the graph has a negative weight cycles. In algorithm 3, the step 1 can be processed in a linear time, if-part of the step 2 (Floyd's all shortest path algorithm) takes $O(k^3)$ and else-part of the step 3 can be processed in a linear time where k is a number of node in G.

5.2 Strategies for Intensional Answers

In Section 3.2, we proposed structural rules and subclassing rules for class hierarchy. Before we get the intensional answers, first of all some rule transformations should be done in order to get unique intensional literals, secondly recursion in subclassing rules should be removed, and "IS_A"-rules to the first order logic should be changed.

Unique intensional literals are literals that are either extensionally or intensionally defined but not both. So if we have literal p which is both EDB-defined and IDB-defined, then rename the extensional literal p^* and introduce a new rule $p \leftarrow p^*$ in the IDB. In doing so, we can handle complete queries as well as incomplete queries for the intensional query processing.

Since structural rules and subclassing rules are conjunction in IDB, we can remove recursion in subclassing rules.

Finally we can change "IS_A"-rules to the first-order logic since the semantic of "IS_A" is implication. For example, $IS\_A(X, Y)$ can be changed $Y \leftarrow X$. Now, IDB corresponds to a set of non-recursive Horn clauses.

The following algorithm will compute intensional answers from a set of non-recursive Horn clauses consisting of $EDB \cup IDB$ and a query $Q(X)$.

Input : A set of non-recursive Horn clauses consisting of $EDB \cup IDB$ and a query $Q(X)$ where X is a tuple of free variables.

Output : A set of $ANS_I(X)$ of intensional answers.

1. Negate the query and convert it into the clause form
2. Repeat for all branches of a resolution tree
   2.1 Perform resolution using subclassing rules, structural rules or new rules
   2.2 If resolvent contains extensional literal p^* then
       If base literal is not in the attributes of p^* then
           current branch is fail branch and return
       else
           current branch is success branch and return
   else /* if resolvent does not contain extensional literal */
       If resolvent contains at least two base literals then
           If factoring is impossible among base literals then
               current branch is fail branch and return
           else
               begin
               Perform the algorithm 3
               If result is FALSE then
                   current branch is fail branch and return
               else if result is FALSE then
                   current branch is success branch and return
                   $ANS_I(X) =$ *selected predicate*
               else
                   current branch is success branch and return
                   $ANS_I(X) =$ *selected predicate* $\wedge$ simplified *predicate*
   Until it cannot be further resolved
3. If all success branches contain extensional answers then
       begin
       choose the highest success branch
       generate the intensional answers by negating resolvent
       end
   else
       begin
       ignore success branch containing extensional answers
       return intensional answers $ANS_I(X)$
       end

(Algorithm 4)

5.3 Example

To show the application of the algorithm introduced in the above section, we will use our example database given in (Figure 1).

EDB and IDB schema looks as follows :

• EDB

vehicle(id, weight, color)
automobile(id, weight, color, size, gas_mileage, no_doors)
family_car(id, weight, color, size, gas_mileage, no_doors, no_seats)
sportscar(id, weight, color, size, gas_mileage, no_doors, speed)
high_speed_car(id, weight, color, size, gas_mileage, no_doors, speed)
normal_speed_car(id, weight, color, size, gas_mileage, no_doors, speed)
water_vehicle(id, weight, color, level, speed)
ship(id, weight, color, level, speed, size)
high_speed_ship(id, weight, color, level, speed, size)
normal_speed_ship(id, weight, color, level, speed, size)

• IDB

"IS_A"-rules
subclassing rules

Frist we rename the extensional literal, add new rules in the IDB, remove recursion, and change "IS_A"-rules to the first order logic :

• EDB

vehicle*(id, weight, color)
automobile*(id, weight, color, size, gas_mileage, no_doors)
family_car*(id, weight, color, size, gas_mileage, no_doors, no_seats)
sportscar*(id, weight, color, size, gas_mileage, no_doors, speed)
high_speed_car*(id, weight, color, size, gas_mileage, no_doors, speed)
normal_speed_car*(id, weight, color, size, gas_mileage, no_doors, speed)
water_vehicle*(id, weight, color, level, speed)
ship*(id, weight, color, level, speed, size)
high_speed_ship*(id, weight, color, level, speed, size)
normal_speed_ship*(id, weight, color, level, speed, size)

• IDB

- "IS_A"-rules

vehicle(X) ← automobile(X)
vehicle(X) ← watervehicle(X)
automobile(X) ← sportscar(X)
automobile(X) ← family_car(X)

watervehicle(X) ← ship(X)
sportscar(X) ← high_speed_car(X)
sportscar(X) ← normal_speed_car(X)
ship(X) ← high_speed_ship(X)
ship(X) ← normal_speed_ship(X)

• subclassing rules

high_speed_car(X) ← speed(X, Y) ∧ greater(Y, 200)
normal_speed_car(X) ← speed(X, Y) ∧ greater(Y, 100) ∧ less(Y, 200)
sportscar(X) ← no_doors(X, Y) ∧ equal(Y, 2)
family_car(X) ← no_doors(X, Y) ∧ equal(Y, 4)
high_speed_ship(X) ← speed(X, Y) ∧ greater(Y, 50)
normal_speed_ship(X) ← speed(X, Y) ∧ lesseq(Y, 50)

• new rules

vehicle(X) ← vehicle*(X)
automobile(X) ← automobile*(X)
family_car(X) ← family_car*(X)
sportscar(X) ← sportscar*(X)
high_speed_car(X) ← high_speed_car*(X)
normal_speed_car(X) ← normal_speed_car*(X)

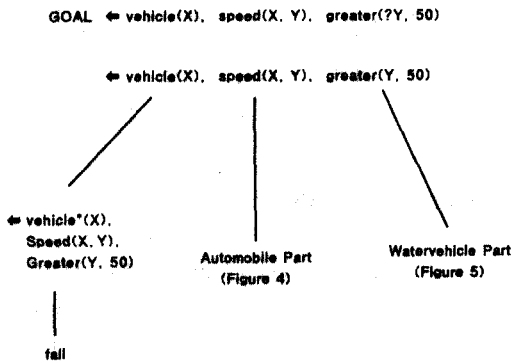
Now let us consider the query "Find a set of vehicle ids where the speed of the vehicle is greater than 50 miles per hour". The query can be written us

$Q(X)$ = vehicle(X) ∧ speed(X, Y) ∧ greater(Y, 50).

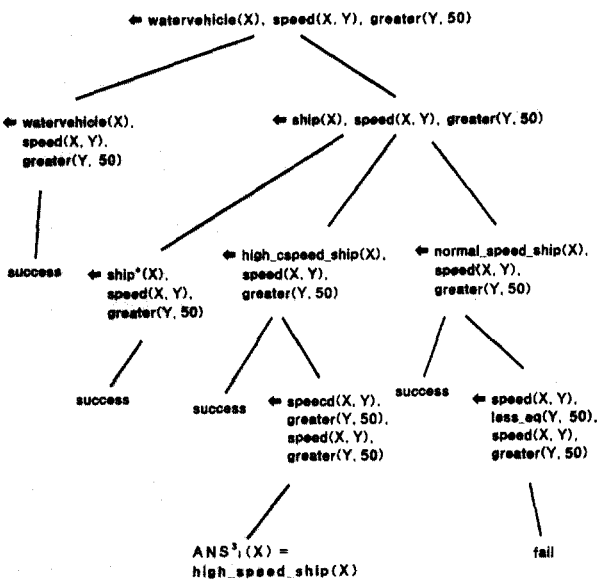
Thus the goal clause is

← vehicle(X), speed(X, Y), greater(Y, 50).

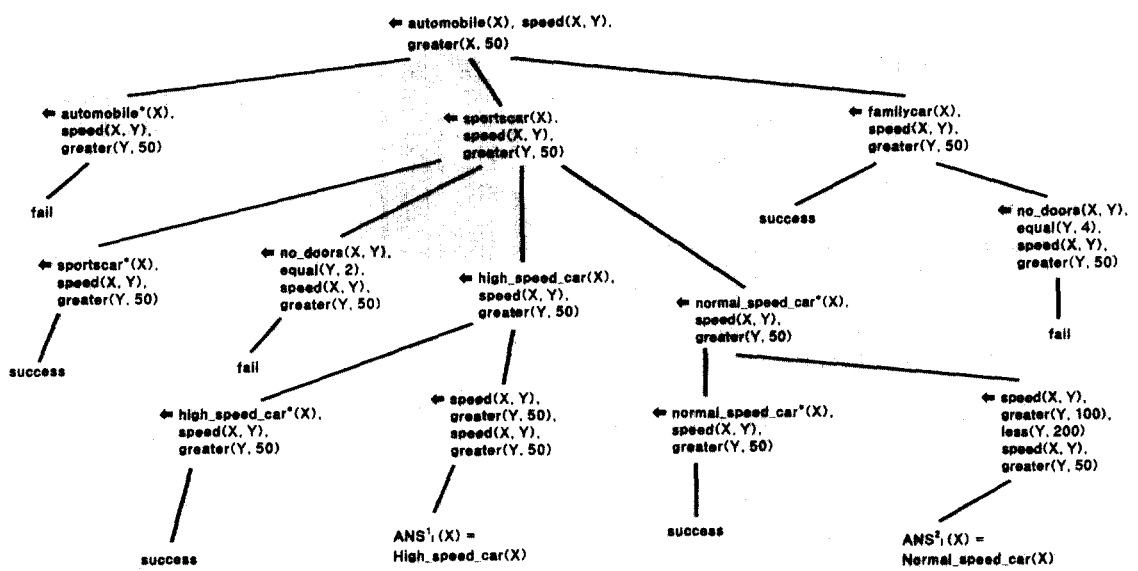
Now the resolution tree is as follows.



(Figure 2) Resolution Tree



(Figure 2.2) Watervehicle Part



(Figure 2.1) Automobile Part

Since there are three branches that have intensional answers, we have following intensional answers :

- $ANS_7^1(X)$ = high_speed_car(X)
- $ANS_7^2(X)$ = normal_speed_car(X)
- $ANS_7^3(X)$ = high_speed_ship(X)

6. Conclusions and Remarks

In this paper, we developed a formalism to obtain intensional answers for a class hierarchy model. By introducing rules into the OODB systems and applying the IQP techniques to the OODB systems, we are able to use the advantages of the semantics of OODB schema.

By using rules derived from the schema information, we are able not only to answer incomplete queries without knowing the exact structure of the database but also to express the answer-set abstractly as the names of classes. It provides us better understanding of the answer.

However, the method in this paper also has a disadvantage. For complete queries, the algorithm in this paper is less efficient than current query languages since our method answers a query only after it generates the complete resolution tree by using structural rules, subclassing rules, and new rules.

In this paper, we only obtain intensional answers for a class hierarchy model but not for a class-composition hierarchy. Our method does not seem to be powerful enough to represent a complex object hierarchy. It is probable that we need more powerful logic for reasoning intensional answers on complex objects.

References

[1] F. Bancilon, *Object-Oriented Database Systems*, in Proceedings of Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp.152-162, Austin, Texas, March, 1988.

[2] S. Böttcher, M. Jarke and W. Schmidt, *Adaptive Predicate Management in Database System*, in Proceedings of 12th VLDB Conference, 1986.

[3] C. Chang and R. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York and London, 1973.

[4] L. Cholvy and R. Demolombe, *Querying a RuleBase*, in Proceedings of the first int'l conference on Expect Database Systems, ed. Kerschberg, L., pp.365-371, Charleston, South Carolina, April, 1986.

[5] Parke Godfrey and Jarek Gryz, *Overview of Dynamic Query Evaluation in Intensional Query Optimization*, in Proceedings of the 5th DOOD, Montreux, Switzerland, pp.425-426, December, 1997.

[6] T. Imielinski, *Intelligent Query Answering in Rule Based Systems*, J. of Logic Programming, Vol.4, No.3, pp.229-258, September, 1987. Also appeared as *Transforming Logical Rules by Relational Algebra*, in Proceedings of Foundations of Deductve Database Systems and Logic Programming, ed. Minker, J., pp.338-377, Washington DC, August, 1986.

[7] A. Motro and Q. Yuan, *Querying Database Knowledge*, in Proceedings of ACM SIGMOD, Atlantic City, New Jersey, pp.173-183, May, 1990.

[8] A. Motro, *Using Integrity Constraints to Provide Intensional Answers to Relational Queries*, in Proceedings of 15th VLDB Conference, 1989.

[9] E. Pascual and L. Cholvy, *Answering Queries Addressed to the Rulebase of a Deductive Database*, in Proceedings of 2nd Int'l Conference on Information Processing and Management of Uncertainty in Knowledgebased Systems, pp.138-145, Urbino, Italy, July, 1988, Springer-Verlag, Lecture Notes in Computer Sciences 313.

[10] A. Pirotte and D. Roelants, *Constraints for Improving the Generation of Intensional Answers in a Deductive Database*, International Conference on Data Engineering, pp.652-659, 1989.

[11] D. J. Rosenkrantz and M. B. Hunt, *Processing conjunctive predicates and queries*, in Proceedings of the Sixth International Conference on VLDB, pp.64-74, Montreal, 1980.

[12] I-Y. Song, H-J. Kim and P. Geutner, *Intensional Query Processing : A three step Approach*, in Proceedings of 1990 International Conference on Database and Expect Systems Application, pp.542-549, Vienna, Austria, Aug. 1990.

[13] Suk-Chung Yoon, Il-Yeol Song and E. K. Park, Intelligent Query Answering in Deductive and Object-Oriented Databases, CIKM, pp.244-251, 1994.

[14] S. C. Yoon, I. Y. Song and E. K. Park, *Semantic Query Processing in Object-Oriented Databases Using Deductive Approach*, CIKM, pp.150-157, 1995.

[15] S. C. Yoon, I. Y. Song and E. K. Park, *Intensional query processing using data mining approaches*, in CIKM, pp. 201-208, 1997.

김 양 희

e-mail : yangh-kim@knupe.ac.kr

1982년 이화여자대학교 수학과 졸업(이학사)

1984년 서울대학교 수학과 졸업(이학석사)

1988년 University of Wisconsin - Madison 전자계산학과(이학석사)

1989년 University of Wisconsin - Madison 전자계산학과(박사과정 이수)

1997년 고려대학교 전자공학과 - 컴퓨터 공학전공 졸업(공학박사)

1985년~1989년 University of Wisconsin - Madison 전자계산학과 T.A.

1990년~1998년 수원과학대학 전자계산과 전임강사, 조교수

1998년~2001년 한세대학교 컴퓨터정보통신공학부 전임강사, 조교수

2001년~현재 한국체육대학교 체육학과 조교수

관심분야 : 분산 및 병렬 알고리즘 및 시스템, 객체지향 데이터베이스, 내포적 질의 처리, 공간 데이터베이스, 데이터베이스 보안 등