APPLYING INTENSIONAL QUERY PROCESSING TECHNIQUES TO OBJECT-ORIENTED DATABASE SYSTEMS

Yang Hee Kim
Department of Computer Science, Suwon Industrial College

Hyoung-Joo Kim
Department of Computer Engineering, Seoul National University

ABSTRACT

Conventional database systems generate a set of facts (extensional answers) as an answer for a given query. This also applies to object-oriented databases where a set of objects will be returned.

However, deductive database systems give an opportunity to obtain the answer of a query as a set of formulas (intensional answers). Intensional answers provide users additional insight to the nature of extensional answers and can be computed only using the rules without accessing the database. So, they can be computed much faster and more cheaply than extensional answers.

In this paper, by introducing rules for abstract expression in the object-oriented database systems and by applying the intensional query processing techniques of deductive database systems to 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.

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] [7]. These objects may belong to different classes within a class hierarchy or they may be different complex objects somehow related to each other.

A deductive database is composed of extensional predicates (facts) and intensional predicates (rules). The extensional predicates contain objects of base relations while the intensional predicates contain deduction rules and conditions.

An intensional answer is a formula that states a condition to be satisfied by objects of the extensional answer. Thus, intensional answers are independent of a particular state in the database. Moreover, intentional answers can be computed much faster and more cheaply than extensional answers since in most of the time extensional answers are much larger than intensional answers. Intensional answers also represent the answer to the given query in a more compact way and extensional answers usually change more often than intensional answers. So, intensional answers provide a more stable answer to a query than the extensional answer. Furthermore, since the intensional answer is similar to a query, it can be evaluated like a query and returns a set of objects which are partial extensional answers. 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 intentional query processing (IQP) techniques to OODB systems. Then, it becomes possible to answer incomplete queries by representing OODB schema in terms of rules. So far, all the query languages in OODB systems known to us are not able to give answers to incomplete queries. An incomplete query is a query on the attribute which belongs to a subclass but not the base class. 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 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.

Rules which represent OODB schema in this paper consist of structural rules and subclassing rules. The structural rules in turn consist of "IS_A"-relation 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 a given 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 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 4, we briefly review the definition of intensional answers, formalize methods to derive intensional answers for a class hierarchy model and give a detailed example. In the last section, we give conclusions and some remarks for possible extensions of the method in this paper.

2 RESEARCHES ON INTENSIONAL ANSWERS

There has been done a lot of researches in this area for last several years. While their approaches are different, all have the common goal which is to answer queries with a set of first-order logic formulas rather than a set of facts. But research which integrates this area and OODB was started only recently.

Cholvy and Demolombe [5] are the first who have considered the problem of providing answers to queries which are independent of a particular set of facts, that is, answers which are valid in all the states associated to these sets of facts. In this case, the answer is a set of first-order logic formulas, defining the rules that a given object should satisfy to belong to the answer. They make the hypothesis that the rule base is not recursive, but their algorithm accepts any form of clauses. In their work, a method is based on Resolution Inference Rules.

Based on the research presented by Cholvy and Demolombe [5], Pascual and Cholvy [10] restrict types of rules in intentional database (IDB) to Horn clauses, because Horn clauses have a lot of advantages. But, they do not give the algorithm to remove redundant resolution step and meaningless intensional answers. Song [13] and Song and Dubin [14] use only Horn-clauses for intensional databases and they give the detailed SLD-resolution steps to avoid generating meaningless intensional answers.

Imielinski [6], Pirotte and Roelants [11], Motro [9] and Motro and Yuan [8] use different approaches. Imielinski [6] introduces the type of answers which mix both atomic facts and general rules from projection, selection and join. So he tries to incorporate intensional predicates as a part of answers. Pirotte and Roelants [11] show how intensional answers can be generated from the query and how integrity constraints can filter out inadequate answers and produce simpler and more informatic answers.

Motro [9] describe a method that applies database constraints to generate intensional answers and Motro and Yuan [8] provide a simple query language incorporating intensional queries.

Vadaparty and Badrinath [16] develop a formalism to represent answers abstractly in OODB. They use the classes in the taxonomy to express the answer-set abstractly and obtain a unique optimal abstract representation.

3 LOGICAL REPRESENTATION OF OBJECT-ORIENTED DATA MODELS

3.1 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.

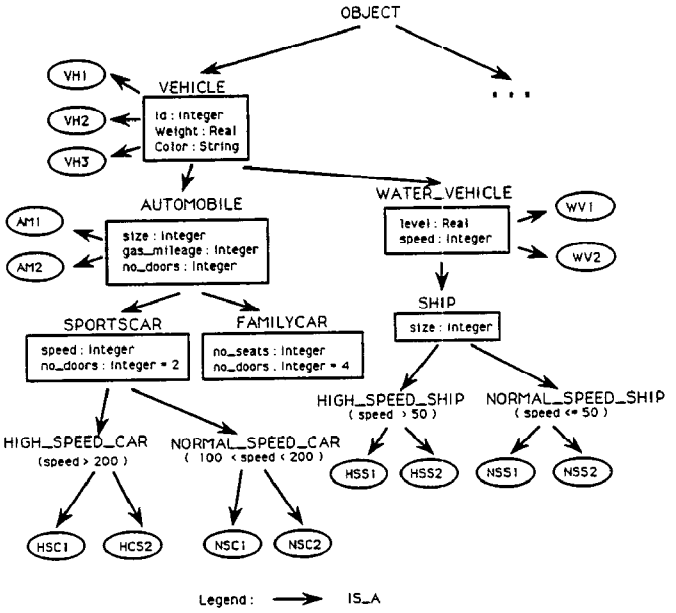

Figure 1. Class Hierarchy

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 systems, 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 but at the same time powerful enough to use to fully account 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 objects 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 for abstract expression

By introducing the notion of rules we can distinguish between different kinds of rules:

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

First, an OODB 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 in 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 OODB system with rules as in deductive database systems.

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 OODB system. These rules come out of the class hierarchy of the objects and the information which is stored in the database schema. The rules concerning the class hierarchy are the structural rules and rules concerning the structual information of the subclassing schema are the subclassing rules.

The structural rules and subclassing rules in Figure 1 are 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) $\leftarrow$ sportscar(X) $\wedge$ speed(X,Y) $\wedge$ greater(Y,200)
  - normal_speed_car(X) $\leftarrow$ sportscar(X) $\wedge$ speed(X,Y) $\wedge$ greater(Y,100) $\wedge$ less(Y,200)
  - sportscar(X) $\leftarrow$ automobile(X) $\wedge$ no_doors(X,Y) $\wedge$ equal(Y,2)
  - family_car(X) $\leftarrow$ automobile(X) $\wedge$ no_doors(X,Y) $\wedge$ equal(Y,4)
  - high_speed_ship(X) $\leftarrow$ ship(X) $\wedge$ speed(X,Y) $\wedge$ greater(Y,50)
  - normal_speed_ship(X) $\leftarrow$ ship(X) $\wedge$ speed(X,Y) $\wedge$ lesseq(Y,50)

4 FORMALIZATION OF INTENSIONAL QUERY PROCESSING IN OBJECT-ORIENTED DATABASE SYSTEMS

4.1 Definition of Intensional Answers

First of all, we briefly review the definition of intensional answers given in [5]. 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, we want to find all the formulas $ans(X)$, such that

$$T \vdash \forall X(ans(X) \rightarrow Q(X))$$

i.e., $\forall X(ans(X) \rightarrow Q(X))$ is a theorem of T. So, the intensional answer $ANS(Q)$ to the query $Q(X)$ is defined by the set of formulas :

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

where $ans(X)$ is a literal.

However, we want to restrict the answers within a defined domain of interest. Let $DP = \{P_1, ..., P_n\}$ be a set of predicate symbols either of the IDB or the extentional database (EDB). And let $L(DP)$ be the first order language whose predicate symbols are $P_1, ..., P_n$. Then define an intensional answer $ANS(Q, DP)$ to a query $Q(X)$ by :

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

To derive an intensional answer, the resolution will be applied to the database theory T and the negation of the theorem $\forall X(ans(X) \rightarrow Q(X))$. But the negation of the thorems is

$$\text{not } (\forall X(ans(X) \rightarrow Q(X)))$$

which is equal to

$$\exists X(ans(X) \wedge not(Q(X)))$$

That leads to the standard form

$$\{ans(X_0), \neg(Q(X_0))\}$$

where X_0 is a tuple of Skolem constants. So, the clause form for the resolution becomes

$$S \cup \{ans(X_0), \neg(Q(X_0))\}$$

where S is a set of clauses transformed from T. Since $ans(X_0)$ is not known at the beginning of the resolution process, we will start with $S \cup \{\neg Q(X_0)\}$. 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 $\neg R(X_0)$.

In the following sections, 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.

4.2 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" [7]. The BNF of simple predicates abbreviated by SP is as follows.

⟨SP⟩ ::= ⟨SP⟩ ∧ ⟨SP⟩ | ⟨SP⟩ ∨ ⟨SP⟩ | ¬⟨SP⟩ | ⟨predicates⟩
⟨predicates⟩ ::= ⟨comparison operator⟩ (⟨variable name⟩, ⟨constant⟩)
| ⟨comparison operator⟩ (⟨variable name⟩, ⟨variable name⟩)
| ⟨comparison operator⟩ (⟨variable name⟩, ⟨variable name⟩ + ⟨ constant⟩)
⟨comparison operator⟩ ::= equal | not_equal | greater | greater_eq | less | less_eq

Rosencrantz and Hunt showed that the satisfiability problem of the set of simple predicate is NP-hard [12]. 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 [12] [7].

Algorithm 1

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 $-c$.

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 conjuntive less_eq predicate (not_equal free predicate that contains only less_eq)

Algorithm 2

Input : A weighted directed graph G with no multiple edges.
Output : A conjuntive 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 $0 \le v_1 + c$.

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

Algorithm 3

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

Output: Truth constant (TRUE or 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 conjective less_eq predicate P
change new comparison predicate to less_eq predicate using step1 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

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 paths 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 nodes in G.

4.3 An algorithm 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 subcalssing 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 conjuction 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).

Algorithm 4

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*

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 TRUE 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
end

Until it cannot be futher 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 and
return intensional answers $ANS_I(X)$
end

4.4 example

To show the application of the algorithm introduced in the above section, we will use our example database given in Fiqure 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

First 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 as

$$Q(X) = \text{vehicle}(X) \wedge \text{speed}(X,Y) \wedge \text{greater}(Y,50).$$

Thus the goal clause is

$$\leftarrow \text{vehicle}(X), \text{speed}(X,Y), \text{greater}(Y,50)$$

Now the resolution tree is as follows.

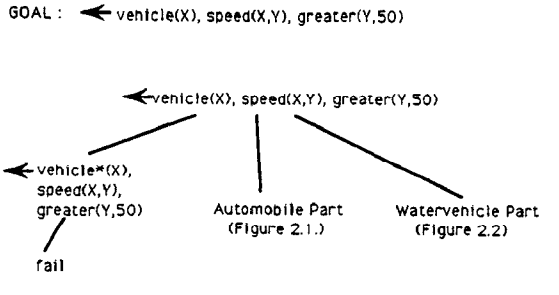

Figure 2. Resolution Tree

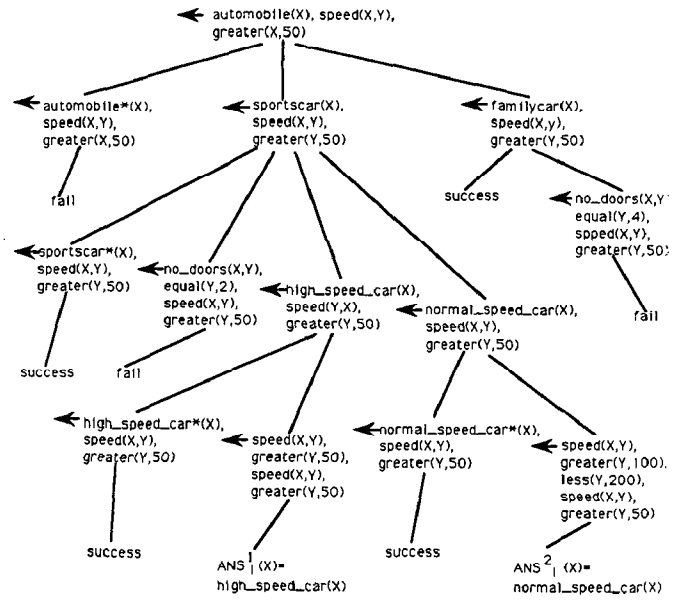

Figure 2.1. Automobile Part

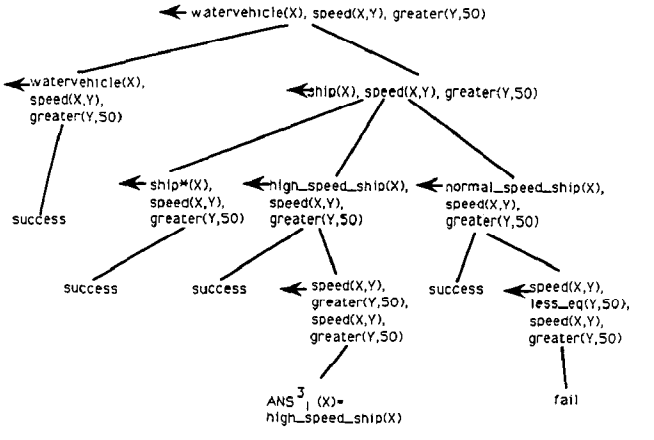

Figure 2.2. Watervehicle Part

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

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

5 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 21-23 1988.

[2] J. Banerjee, et al., *Data Model Issues on Object-Oriented Applications*, in ACM Transactions on Office Information Systems, Vol.5, No.1, March, 1987.

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

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

[5] L. Cholvy and R. Demolombe, *Querying a RuleBase*, in Proceedings of the first int'l conference on Expert Database Systems, ed.Kerschberg, L., pp. 365-371, Charleston, South Carolina, April 1-4, 1986.

[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 Deductive Database Systems and Logic Programming, ed. Minker, J., pp. 338-377, Washington DC, August 8-12, 1986.

[7] H. J. Kim, *Logic Design of Object-Oriented Database Schema*, Technical Report : GIT-ICS-89/06, January, 1989.

[8] A. Motro and Q. Yuan, *Querying DataBase Knowledge*, Proceedings of ACM SIGMOD, Atlantic City, New Jersey, May 23-25, 1990, pp. 173-183.

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

[10] 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.

[11] A. Pirotte and D. Roelants, *Constraints for Improving the Generation of Intensional Answers in a Deductive Databases*, 1989 Int'l Conference on Data Engineering, pp. 652-659.

[12] 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.

[13] Song, I-Y., Intensional Query Processing in Deductive Database Systems, Ph.D Dissertation, Louisiana State University, August 1988.

[14] I-Y. Song and D.Dubin, *IQPS:Implementation of an Intensional Query Processing Shell*, submitted for publication.

[15] 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 Expert Systems Application, pp. 542-549, Vienna, Austria, Aug.29-31, 1990.

[16] K. Vadaparty and B. R. Badrinath, *User-controlled Abstract Answers in Object-Orieneted Systems*, Technical Report : LCSR-TR-164, Department of Computer Science, Rutgers University, Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, April, 1991.