Semantic Query Processing in Object-Oriented Databases Using Deductive Approach

S. C. Yoon
Dept. of Computer Science
Widener University
Chester, PA 19013

I. Y. Song
College of Information Studies
Drexel University
Philadelphia, PA 19104

E. K. Park
Dept. of Computer Science
U.S. Naval Academy
Annapolis, MD 21402

Abstract

In this paper, we present a method to utilize semantic constraints that play an important role in search space reduction and termination of query evaluation in object-oriented databases. Our approach consists of three successive refinement steps: rule generation, semantic knowledge compilation, and semantic reformulation. In the rule generation step, we generate a set of deductive rules based on an object-oriented database schema or semantic knowledge about the domain of the database. In the semantic knowledge compilation step, we compile semantic knowledge together with object-oriented database schema to identify semantic knowledge that is potentially relevant(beneficial) to each class in the object-oriented database schema. We associate the fragments of valid and useful semantic knowledge, called *residues* with classes. After this step, semantic knowledge is grouped according to the classes that it references. During the semantic reformulation step, we receive a user's query, select the set of relevant semantic knowledge in a query context, and transform the query with associated semantic knowledge into another form which is more efficiently processed. To our knowledge, there is no significant research that has been done about semantic query optimization in object-oriented databases using deductive approach. The unique contribution of this paper is that we extend semantic query optimization techniques developed for deductive databases to apply to object-oriented databases. Our approach attempts to minimize the number of operations that will be performed at run time.

1 Introduction

Object-oriented databases are currently receiving a lot of attention from both the experimental and the theoretical standpoint[1, 11, 23]. Object-oriented databases with behavioral encapsulation support powerful data abstractions, but are generally slow in processing a query because they handle huge search space to manipulate complex data objects. On the other hand, relational databases provide efficient query processing, but are not sufficiently powerful to support new application domains because they have flat representation of data. There is a need to develop key techniques that will allow object-oriented databases to attain performance comparable to relational databases in query processing, thus making such databases viable alternative for managing large data volumes.

When processing a query in object-oriented databases, current optimization techniques mainly rely upon the syntactic knowledge and storage details. They do not utilize meaningfully the properties of an application domain which are likely to play a key role during optimization process. Current object-oriented databases do not take advantages of semantic information during query optimization and processing. However, this situation can be different in relational databases and deductive databases. Several semantic query processing algorithms and heuristics have been proposed in those databases [3, 5, 7, 8, 9, 10, 12, 14, 16, 17, 18, 19, 20, 21, 22]. In both databases, semantic knowledge is used to generate semantically equivalent queries, which can be executed more efficiently over the database than the original query by reducing as much as possible the size of the data that must be handled.

In this paper, we are concerned with the use of domain specific semantic knowledge to answer queries over object-oriented databases in an efficient manner. We present a strategy to optimize a query in object-oriented databases by transforming the original query into another semantically equivalent one such that the transformed query can be evaluated more efficiently. This paper describes a scheme to utilize semantic knowledge in optimizing a user specified query. The scheme introduces semantic rules for semantic knowledge and compiles those rules together with object-oriented database schema. The scheme

CIKM '95, Baltimore MD USA
© 1995 ACM 0-89791-812-6/95/11..$3.50

yields an equivalent, but potentially more profitable, form of the original query.

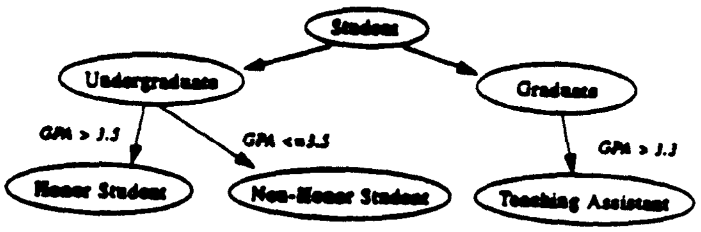
We divide our approach into three successive refinement steps: rule generation, semantic knowledge compilation, and semantic reformulation. In the rule generation step, we generate a set of deductive rules based on an object-oriented database schema or semantic knowledge about the domain of the database. In the semantic knowledge compilation step, we identify semantic knowledge that is potentially relevant(beneficial) to each class. We associate the set of valid and useful semantic knowledge with each class. Semantic knowledge is grouped according to the classes that it references. During the semantic reformulation step, we receive a user's query, select the set of relevant semantic knowledge in a query context, and transform the query with associated semantic knowledge into another form which is more efficiently processed.

To our knowledge, there is no significant research that has been done about semantic query optimization in object-oriented databases using deductive approach. The unique contribution of this paper is that we extend semantic query optimization techniques developed for deductive databases to apply to object-oriented databases. In particular, we suggest a method to associate semantic knowledge with relevant classes in an object-oriented database schema in an efficient manner. In our approach, the rule generation step and the semantic knowledge compilation step are independent of queries posed to a database and hence are computed once prior to the processing of any query. Our approach attempts to minimize the number of operations that will be performed at run time. We consider our work as a first step in semantic query processing in object-oriented databases.

This paper is structured as follows. Section 2 introduces examples to show the advantages of semantic query optimization. Section 3 recalls a brief background on object-oriented databases and deductive databases. Section 4 surveys related works. Section 5 presents our approach. Section 6 discusses possible extensions of our method.

2 Motivating Example

The following example illustrates the advantages of semantic query optimization. In this example, we only consider a generalization hierarchy of classes which is an essential component of every semantically rich object-oriented database model. The hierarchy helps in factoring out shared specifications and implementation in application.
Suppose we have the following object-oriented database schema about a portion of a university.



The following is sample semantic knowledge that exists in the above database: "The lowest acceptable GPA for an undergraduate student is 2.0 and the lowest acceptable GPA for a graduate student is 3.0". Now we have a query asking "find all graduate students whose GPA is lower than 3.0". The query will certainly fail because of the above semantic knowledge. If we use the semantic knowledge, we can easily detect a contradiction. So, we don't have to process the query further. In this case, semantic knowledge is used to prevent the exploration of search space that is bound to fail.

From the above database, we know that Honor Student is a subclass of Undergraduate with a more specific constraints– GPA is above 3.5. Now suppose we have a query asking "who are undergraduate students whose GPA is above 3.5?". In a conventional object-oriented database, we need to search through all the subclasses of Undergraduate to find answers. If a database can process semantic knowledge, then we only need to search the subclass Honor Student to find all objects that satisfy a query. We don't have to search the other subclass such as Non-Honor Student which is unproductive. In this case, semantic knowledge is used to cut down the amount of computation that must be done at run time.

As shown in the above example, we are able to reduce query processing time in semantically rich object-oriented databases if we can use available semantic knowledge efficiently about the database. Semantic knowledge can be a powerful tool for answering queries. Using semantic knowledge to process a query can reduce search space and response time.

The primary benefits of semantic query processing are:

1. Detection of unsatisfiable conditions: it is possible to determine that a query does not have an answer when the existence of the answer would violate semantic knowledge.
2. Restriction of search space: it is possible to use a partial search by transforming a query into a se-

mantically equivalent query that can be answered more efficiently.

3. Answering a query: it may be possible to answer a query without accessing the database if there exists semantic knowledge to provide a unique answer.

When a query is evaluated, we can make deduction in a more intelligent way and avoid wasting time trying to solve semantically meaningless queries.

3 Object-Oriented Databases and Deductive Databases

The integration of programming concepts with database is one of the most significant advances in the evolution of database systems. This approach has developed two popular models, object-oriented databases and deductive databases. In this section, we review the object-oriented database model and the deductive database model, primarily derived from [6, 11]. The goal of objective-oriented databases is to apply object-oriented concepts in object-oriented programmings in modeling data. Object-oriented databases combine the data abstraction and computation models of object-oriented languages with the performance and consistency features of databases. The goal of deductive databases is to integrate rules and traditional database of facts in logic programming to support complex reasoning.

Object-oriented databases have some features that do not exist in a conventional database. An object-oriented database is a set of object-oriented concepts, including object-identity, encapsulation, inheritance, and polymorphism, for modeling data. These concepts are sufficiently powerful to support data-modeling requirements of many types of application. In object-oriented database systems, objects sharing structure(type) and behavior(methods) are grouped into classes and objects are organized in a hierarchy of classes and subclasses. The class hierarchy captures the relationship between a class and its subclass. Subclasses inherit all the attributes of their superclasses and can have some of their own attributes. Object at any level of the hierarchy inherits all the properties of object higher up in the hierarchy. The access scope of a given query on a class, say C, is either the set of instances of C, or the set of instances of the entire class hierarchy rooted at C(that is, all subclasses of C). The result of the query may be the set of objects that belong to different classes within a class hierarchy.

Deductive databases have some features that do not exist in a conventional database. A deductive database is a database in which new facts may be derived from the facts that are explicitly stored by using an inference system. A deductive database comprises an extensional database(EDB) consisting of a set of facts explicitly stored in a physical database, and an intensional database(IDB) consisting of a set of deductive rules. These rules can be used to derive new facts from the facts in the EDB. In a deductive database, there is one advantage: facts, deduction rules and queries can be written in a uniform database language typically based on a first-order logic. A rule has the form(Prolog-like notation[4, 15]) $a : -b_1, b_2, \quad b_m$ m ≥ 0 where a is an atomic formula and b_i's are literals. All the variables occuring in the rule are assumed to be universally quantified. The literal a is called the *head* of the rule, the b_i's are referred to as the *body* of the rule. Every rule can be represented as a clause that is finite disjunction of one or more literals. So, our rule can be represented as a clause, and would look like:

$$a \bigvee \neg b_1 \bigvee \neg b_2 \bigvee \ldots \bigvee \neg b_m$$

4 Related Works

Semantic query optimization can be regarded as the process of transforming a query into an equivalent form, which can be evaluated efficiently. There have been some interesting studies on semantic query optimization in relational and deductive databases. The main idea of the studies is that integrity constraints can be utilized to optimize user queries. King[12] describes an algorithm which uses a set of transformation heuristics. These heuristics help to limit the number of transformations by specifying how each heuristic can be used to transform a given query. Xu[20] presents heuristics similar to King's, adding a control strategy for selecting appropriate transformations. Hammer and Zdonik[7] describe how a system can use a knowledge base to perform semantic query optimization. Jarke et al.[10] describe a graph-theoretic approach to semantic query simplication implemented in Prolog. Similarly, Shenoy and Ozsoyoglu[17] suggest a graph-theoretic approach to achieve semantic query optimization by identifying redundant conditions and eliminating them from the query graph. Chakravarthy et al.[3] describe a method for using theorem-proving techniques to compile the semantic constraints with an IDB of the system and derive all possible query transformations. But, their approach does not define methods to select useful transformation from the set of transformations. Yoon and Henschen[21, 22] extend Chakravarthy's work by introducing control strategies to find useful transformations.

The entire field of semantic query processing in object-oriented databases is still evolving rapidly. To the best of our knowledge, there is no work done about semantic query optimization in object-oriented databases using deductive approach. We consider our work as a first step in semantic query processing in object-oriented databases.

5 Our Approach

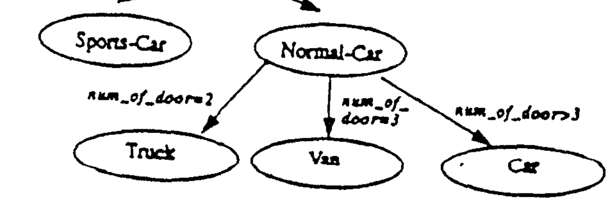
Our approach consists of three steps: rule generation, semantic knowledge compilation, and semantic reformulation. The three steps are partitioned into two categories: processing that can be done statically once and processing that has to be performed at run time. The first two steps belong to the first category and the last step belongs to the second category. So, the rule generation step and the semantic knowledge compilation step are independent of queries posed to a database and hence are computed once prior to the processing of any query. Our approach attempts to minimize the number of operations that will be performed at run time. In this section, we explain those three steps.

5.1 Rule Generation

In this phase, we perform three steps. In the first step, we generate two sets of deductive rules that are represented in the form of non-recursive Horn clause. The first set is based on the class hierachy in an object-oriented database schema. In the first set, we focus on inclusion inheritance that a class C' is a subclass of a class C. We convert an object-oriented database schema into a set of non-recurisive Horn clauses. The second set is based on semantic knowledge in the object-oriented database domain. In the second set, we include constraint inheritance that the class C' must satisfy a given constraint.

We can classify rules into two different categories: rules to specify the class hierarchy and rules to specify semantic knowledge. Semantic knowledge can be further classified into semantic knowledge within a class or semantic knowledge across classes.

Suppose that the figure illustrates a portion of an automobile and its types.



The rules in the first category represent the hierarchy between a class and its superclass. In the above example, there is a class Automobile having automobile types, Sportscar and Normal-Car as subclasses. Then we have the rules:

Automobile(x):- Sports-Car(x)
Automobile(x):- Normal-Car(x)

where x is a variable vector to represent a set of attributes.

In the above example, we have the following semantic knowledge: "If the number of cylinders in an automobile is greater than 4, then the automobile belongs to a subclass Sports-Car". This semantic knowledge involves two classes of Automobile and Sports-Car. In this case, we may need to introduce a new literal called *property literal*. The format of a property literal is P(x,y) where P is a predicate name and x represents an object that belongs to a class and y represents a property of the object x. We represent the knowledge as follows:

Sports-Car(x):- Automobile(x),
Num-of-Cylinder(x,y),GT(y,4)

The literal Num-of-Cylinder is a property literal to show that the automobile x has y cylinders and the literal GT is a comparison literal to mean "greater than".

We have another semantic knowledge, saying that a Normal-Car whose number of doors is equal to 2 is a truck. We can represent the above knowledge as follows:

Truck(x):- Normal-Car(x),Num-of-Door(x,y),EQ(y,2)

The literal Num-of-Door is a property literal to show that the Normal-Car x has y doors and the literal EQ is a comparison literal to mean "equal to".

Semantic knowledge can be represented in the form:

$$H{:}\text{-}\ B_1, \ldots, B_n, P_1, \ldots, P_m, E_1, \ldots, E_k$$

where H and each B_i, $1 \le i \le n$, are predicates to represent classes, each P_j, $1 \le j \le m$, is a property literal, and each E_l, $1 \le l \le k$, is a comparison literal.

In the second step, we introduce an EDB literal for each class appeared in the object-oriented schema to represent the objects that belong to each class. For example, each object that belongs to the class Automobile can be represented with the EDB literal Automobile such as Automobile(x).

In the third step, some rule transformations are done in order to get unique intensional literals. *Unique intensional literals* mean that a literal should either be extensionally or intensionally defined but not both. One can always get rid of this equality of names by renaming the extensional literal to p^* and introducing the rule p:- p^*. In the above example, the literal Automobile is both extensionally and intensionally defined.

So, we rename the extensional literal to *Automobile** and introduce the rule

Automobile(x):- *Automobile**(*x*)

5.2 Semantic Knowledge Compilation

In this section, we present a method of compiling semantic knowledge. Semantic knowledge compilation is performed at compile time in order to minimize the amount of computation that must be done at run time. Any remaining optimization is done at the semantic reformulation step. The main goal of this step is to associate semantic knowledge fragment with relevant classes. Semantic knowledge fragment is called *residue*. At the semantic reformulation step, we consider only the group of residues attached to the class that appear in a query. A modified version of the subsumption technique in theorem proving, partial subsumption[3], provides a procedure to determine if semantic knowledge is relevant to a class. Once the semantic knowledge compilation has been performed, semantic knowledge is no longer needed for query processing because all the information is retained in the constraint list of each class.

Definition: A clause C *subsumes* a clause D if and only if there is a substitution σ such that $C\sigma \subseteq D$. D is called the subsumed clause[2].

Definition: Semantic knowledge A *partially subsumes* a class C if and only if there is a set of substitutions $\sigma_1, \sigma_2, \ldots, \sigma_n$ and a set of clauses $A_1, A_2, \ldots, A_n$ where each A_i, i=0 to n, is a subset of A and each $A_i\sigma_i$ is a subset of C, but there is no substitution γ such that $A\gamma$ is a subset of C. That is, semantic knowledge A partially subsumes a class C if a subset of A subsumes C but A does not subsume C.

Definition: Let E be a wff and $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ be a substitution. Then $E\theta^{-1}$ is an expression obtained by replacing simultaneously each occurrence of the term t_i, $1 \leq i \leq n$, in E by the variable v_i. The expression $E\theta^{-1}$ is called back substitution of θ in the wff E[2].

Definition: A *residue* of semantic knowledge A with respect to a class C, is defined as the set of literals $((A - A_i)\sigma_i)\theta^{-1}$ for each A_i which is a subset of A and subsumes C.

If semantic knowledge A partially subsumes a class C, then a residue is generated. Residues generated in this phase will be used in the semantic reformulation step. The following algorithm explains briefly the process of generating residues.

```
Algorithm    Residue Generation
begin
 for each class C do
   for each semantic knowledge A do
     if A does not subsumes C then
       begin
         for each $A_i$ which is a subset of A do
           begin
             if $A_i$ particially subsumes C then
               begin
                 let $\sigma$ be the substitution for the partial
                 subsumption
                 A residue $(A-A_i)\sigma$ is generated
               end
           end
       end
end.
```

Once a set of residues is found for a class, each residue in the set is incorporated into the class. That is, each class can be semantically constrained with the list of its own residues. Each class attached with residues has the form, C $\{R_1, \ldots, R_m\}$ where each R_j, $1 \leq j \leq m$, is a residue obtained from the class C and semantic knowledge. Each residue does not enable the deduction of new answers but rather give information about existing knowledge and answers.

After applying partial subsumption technique directly between the set of semantic knowledge and each class, the portion of each semantic knowledge is attached to the corresponding class in which the semantic knowledge is relevant. We show how residues are generated.

Example: We have a class Automobile and the following semantic knowledge, s1 and s2:

Sports-Car(x):- Automobile(x),
    Num-of-Cylinder(x,y),GT(y,4) (s1)

Normal-Car(x):- Automobile(x),
    Num-of-Cylinder(x,y),LE(y,4) (s2)

After the system performed the partial subsumption process between the class Automobile and the semantic knowledge, the following two *residues* are obtained.

Sports-Car(x):- Num-of-Cylinder(x,y),GT(y,4)
    (from Automobile and s1)

Normal-Car(x):- Num-of-Cylinder(x,y),LE(y,4)
    (from Automobile and s2)

The class Automobile is semantically constrained with those two residues. That is, the class Automobile is attached the residues.

Automobile(x) {
  Sports-Car(x):- Num-of-Cylinder(x,y),GT(y,4)
  Normal-Car(x):- Num-of-Cylinder(x,y),LE(y,4) }

In this phase, we group semantic constraints according to classes they reference. So, we reduce the overhead of retrieving constraints and checking whether each constraint is relevant to a query or not.

5.3 Semantic Reformulation

In this section, we provide a discussion of the usage of semantically constrained classes during query processing. In this phase, we receive a user's query and transform the query into a semantically equivalent one which can be executed more efficiently. The residues generated in the previous step are used to generate semantically equivalent queries. During this stage, users might be able to reduce search space or users might even be able to discard the query itself totally. At this phase, we consider only potentially relevant semantic knowledge instead of all the semantic knowledge ever generated. Among the potentially relevant semantic knowledge selected, only relevant semantic knowledge is finally used and the relevance is decided by a query context. So, our approach is much more efficient and can be easily justified.

When a query is presented to the database system, the system

(1) brings the set of all residues attached to the classes the query references and

(2) select residues to be considered relevant to the query.

Definition: A residue is *relevant* to a query if the body of the residue is evaluated to true.

During this stage, we check if a residue is relevant to a query or not. If the body of a residue evaluates to true, then the residue is relevant to the query. If a residue is relevant to a query, then we can use the head of the relevant residue to transform the query into an equivalent one, which can be evaluated efficiently.

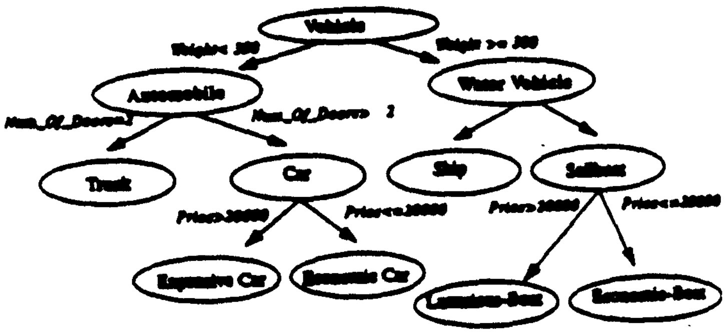
Suppose that we have a query asking "find all automobiles whose number of cylinder is greater than 4". The query can be represented as follows:

:- Automobile(x), Num-of-Cylinder(x,y), GT(y,4)

We have two residues attached to the class Automobile. If we follow the above definition, the first residue is relevant while the second residue is not relevant because the body of the first residue is satisfied by the conditions in the given query while the body of the second residue is not. In this case, if we use the first residue in the class Automobile, we get Sports-Car(x) and replace the query with Sports-Car(x). We only need to search the class Sports-Car to find answers for the given query. We don't have to search the other classes that are bound to fail. The original query can be transformed into a semantically equivalent query which can be processed efficiently.

5.4 An Example

This example shows the application of our approach and algorithm introduced in section 5 Suppose we have the following class hierarchy about vehicles.



In the rule generation phase, we generate a set of deductive rules based on the above object-oriented database schema or semantic knowledge. In the first step, we get the following IDB rules that belong to two different categories.

IDB:

1. Rules to specify the class hierarchy:

Rule 1	Vehicle(x):- Automobile(x)
Rule 2	Vehicle(x):- Watervehicle(x)
Rule 3	Automobile(x):- Truck(x)
Rule 4	Automobile(x):- Car(x)
Rule 5	Watervehicle(x):- Ship(x)
Rule 6	Watervehicle(x):- Sailboat(x)
Rule 7	Car(x):- Expensive-Car(x)
Rule 8	Car(x):- Economic-Car(x)
Rule 9	Sailboat(x):- Luxurious-Boat(x)
Rule 10	Sailboat(x):- Economic-Boat(x)

2. Rules to specify semantic knowledge:

Rule 11	Automobile(x):- Vehicle(x), Weight(x,y),LT(y,300)
Rule 12	Watervehicle(x):- Vehicle(x), Weight(x,y),GE(y,300)
Rule 13	Truck(x):- Automobile(x), Num-Of-Door(x,y), EQ(y,2)
Rule 14	Car(x):- Automobile(x), Num-Of-Door(x,y), GT(y,2)
Rule 15	Expensive-Car(x):- Car(x),Price(x,y), GT(y,20000)
Rule 16	Economic-Car(x):- Car(x),Price(x,y), LE(y,20000)
Rule 17	Luxurious-Boat(x):- Sailboat(x), Price(x,y),GT(y,20000)

Rule 18	Economic-Boat(x):- Sailboat(x), Price(x,y),LE(y,20000)

The comparison literal LT means "less than", LE means "less than and equal to", GT means "greater than", GE means "greater than and equal to".

In the second step, we introduce the following EDB literals to represent objects that belong to each class.

EDB:
Vehicle(x)
Automobile(x)
Watervehicle(x)
Truck(x)
Car(x)
Ship(x)
Sailboat(x)
Expensive-Car(x)
Economic-Car(x)
Luxurious-Boat(x)
Economic-Boat(x)

In the third step, we need to rename the extensional literals to get unique intensional literals. Therefore, we add new rules to the IDB.

New rules:

Rule 19	Vehicle(x) :- *Vehicle**(x)
Rule 20	Automobile(x):- *Automobile**(x)
Rule 21	Watervehicle(x):- *Watervehicle**(x)
Rule 22	Truck(x):- *Truck**(x)
Rule 23	Car(x):- *Car**(x)
Rule 24	Expensive-Car(x):- Expensive-*Car**(x)
Rule 25	Economic-Car(x):- Economic-*Car**(x)
Rule 26	Luxurious-Boat(x):- Luxurious-*Boat**(x)
Rule 27	Economic-Boat(x):- Economic-*Boat**(x)

In the above rules, the predicates denoted with the symbol * represent that those predicates are just EDB predicates. The variable x in each predicate is a variable vector to represent a set of attributes in the predicate.

In the semantic knowledge compilation phase, we use partial subsumtion technique to identify potentially relevant semantic knowledge to each class. In the above example, we get the following residues.

From rule 11 and Vehicle(x),
Automobile(x):- Weight(x,y),LT(y,300) (1)
From rule 12 and Vehicle(x),
Watervehicle(x):- Weight(x,y),GE(y,300) (2)
From rule 13 and Automobile(x),
Truck(x):- Num-of-Door(x,y),EQ(y,2) (3)
From rule 14 and Automobile(x),
Car(x):- Num-of-Door(x,y),GT(y,2) (4)
From rule 15 and Car(x),
Expensive-Car(X):- Price(x,y),GT(y,20000) (5)
From rule 16 and Car(x)
Economic-Car(x).- Price(x,y),LE(y,20000) (6)
From rule 17 and Sailboat(x)
Luxurious-Boat(x):- Price(x,y),GT(y,20000) (7)
From rule 18 and Sailboat(x)
Economic-Boat(x):- Price(x,y),LE(y,20000) (8)

The class Vehicle has the Residues 1 and 2, the class Automobile has 3 and 4, the class Car has 5 and 6, and the class Sailboat has 7 and 8.

In the semantic reformulation phase, suppose that we receive the following query: "find all cars whose price is greater than 20000" in the execution phase. That is,

- Car(x), Price(x,y),GT(y,20000)

In the first step, we check the residues in the class Car. We have two candidates, Residues 5 and 6. That is, we have two potentially relevant residues to the query. The body of the first residue satisfies the given conditions in the query. So, we are able to use the head of the residue, Expensive-Car(x) . That means that answers can be found in the class Expensive-Car(x). The body of the second residue is false. So, we ignore it. That means that we don't have to search the class Economic-Car(x). The first residue is relevant and beneficial to the query while the second residue is not.

6 Conclusion

In this paper, we have presented a method to improve performance in query processing by using semantic knowledge about a given object-oriented database domain. We have showed that a query answering process which considers semantic knowledge could answer the query with a partial search–in some cases with no search because we can identify and eliminate unproductive search activity.

Our approach consists of three steps: rule generation, semantic knowledge compilation, and semantic reformulation. In the rule generation step, we have generated a set of deductive rules based on an object-oriented database schema or semantic knowledge about the domain of the database. In the semantic knowledge compilation step, we have compiled the list of semantic knowledge rules into classes and then have identified semantic knowledge that is potentially relevant(beneficial) to classes. Semantic knowledge is grouped according to the classes that it references. During the semantic reformulation step, we receive a user's query, select the set of relevant semantic knowledge in a query context, and transform the query with associated semantic knowledge into another form which is more efficiently processed.

In most applications, object-oriented databases handle huge search space because they manipulate complex data objects. When processing a query, we need to reduce as much as possible the search space to the portion of a database relevant to the query, which improves search efficiency substantially. Our approach using semantic knowledge has achieved the goal by reducing large amount of data to be the interesting subset of the database suitable for further consideration and processing. The unique contribution of this paper is that we have applied and extended the semantic query optimization methods developed for dedcutive databases to object-oriented databases. Our approach attempts to minimize the number of operations that will be performed at run time.

In near future, it may be interesting to use different computational methodologies besides partial subsumption techniques in associating semantic knowledge with relevant classes. As a next step, we will implement a prototype of a semantic query optimizer. We consider that this paper is the first work in semantic query processing in object-oriented databases.

References

[1] Bancilhon, F. et al., *Building an object-oriented database system*, Morgan Kaufmann Publishers, 1992

[2] Chang, C.L. and Lee, C.T., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973

[3] Charkravarthy, U.S., et.al., "Semantic Query Optimization in Expert Systems and Database Systems", *Proceedings of 1st International Workshop on Expert Database Systems*, 1988

[4] Cloksin, W.F. and Mellish, C.S., *Programming in Prolog*, Springer-Verlag, New York, 1984

[5] Freytag, J. et al., *Query Processing For Advanced Database Systems*, Morgan Kaufmann Publishers, 1994

[6] Gallaire, H., Minker, J., Nicolas, J.M., "Logic and Database: A Deductive Approach", *Computing Surveys*, 1984

[7] Hammer, M.M., Zdonik, S.B., "Knowledge Based Query Processing", *Proceedings of 6th International Conference on VLDB*, 1980

[8] Han, J., Henschen, L.J., and Lu.W., "Search Strategies for Finding Partial Answers in Large Knowledge-Bases", *Proceedings of Symposium on Principles of Database Systems*, 1988

[9] Han, J, "Constraint-based reasoning in deductive databases", *Proceedings of 7th Data Engineering Conference*, 1991

[10] Jark, M., "External Semantic Query Simplification: A Graph- Theoretic Approach and Its Implementation in Prolog", *Proceedings of 1st International Workshop on Expert Database Systems*, 1986

[11] Kim, Won, *Introduction to Object-Oriented Databases*, MIT Press, 1990

[12] King, J., Semantic Query Optimization, *Ph.D. Dissertation*, Dept. of Computer Science, Stanford University, 1981

[13] Kowalski, R., *Logic for Problem Solving*, North-Holland, 1979

[14] Krishnamurthy, R., Boral, H., and Zaniolo, C., "Optimization of Nonrecursive Queries", *Proceedings of 12th International Conference on VLDB*, 1986

[15] Lloyd, J.W., *Foundation of Logic Programming*, Springer-Verlag, 1984

[16] Ohsuga, S., "Knowledge processing and its application to engineering design", *Knowledge Engineering*, Mcgraw-Hill, 1989

[17] Shenoy, S.T., Ozoyuglu, "A system for Semantic Query Optimization", *Proceedings of ACM-SIGMOD Conference Management of Data*, 1987

[18] Siegel, M., "Automatic rule derivation for semantic query optimization", *Technical Report*, Dept. of Computer Science, Boston University, 1987

[19] Ullman, J., *Principles of Database and Knowledge-base Systems* Vol I and II, Computer Science press, 1988

[20] Xu, G.D., "Search control in Semantic Query Optimization", *M.S. Thesis*, Dept. of Computer Science, University of Massachusetts, 1983

[21] Yoon, S.C. and Henschen, L.J., "Intelligent Control in Expert Database Systems", *Proceedings of Fourth IEEE International Symposium on Intelligent Controls*, 1989

[22] Yoon, S.C. and Henschen, L.J., "Search Strategies in Knowledge Base Systems", *Proceedings of Canadian Conference on Electrical and Computer Engineering*, 1990

[23] Zdonik, S. and Maier, D., *Readings in object-oriented database systems*, Morgan Kaufmann Publisher, 1990