Containment and Minimization of Positive Conjunctive Queries in OODB's

(Extended Abstract)

Edward P.F. Chan†
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
epfchan@dragon.waterloo.ca

Abstract

With the availability of high-level declarative query languages in an object-oriented database system (OODB), the burden of choosing an efficient execution plan for a query is transferred from the user to the database system. A natural first step is to use the typing constraints imposed by the schema to transform a query into an equivalent one that logically accesses a minimal set of objects. We propose a class of queries called conjunctive queries for OODB's. A conjunctive query can be expressed as an equivalent union of queries in a special form called terminal conjunctive queries. We first characterize the containment, and hence equivalence, conditions for the class of terminal conjunctive queries. We then study a subclass of conjunctive queries called positive conjunctive queries. We characterize the containment and equivalence conditions, as well as derive an algorithm for finding an exact minimization for the class of positive conjunctive queries. The equivalent minimized query is expressed as a union of terminal positive conjunctive queries with the property that the variable search space is minimal among all the unions of positive conjunctive queries.

†The author is a member of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council of Canada, and the participation of PRECARN Associate Inc.

11th Principles of Database Systems/6/92/San Diego, CA
© 1992 ACM 0-89791-520-8/92/0006/0202...$1.50

1 Introduction

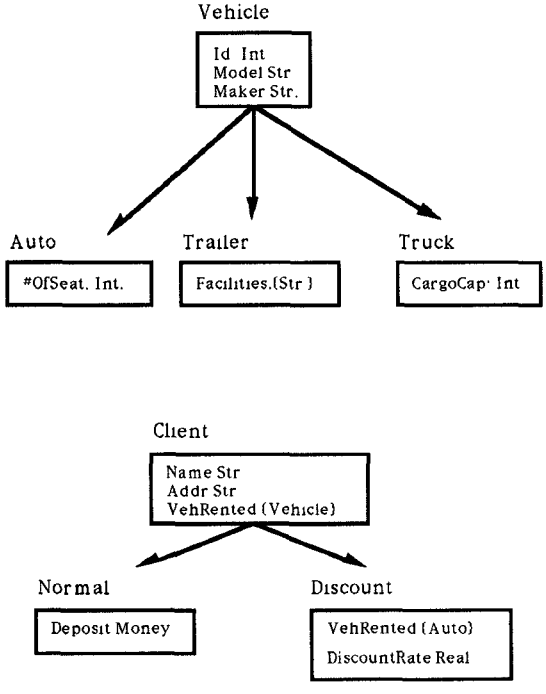
The initial attempts at constructing Object-Oriented Database Systems (OODB's) provided only navigational programming languages for manipulating data [25, 6]. The lack of query languages likes those available in the relational systems has been criticized as a major drawback of the object-oriented approach [32, 4]. Consequently, recent research on OODB's has emphasized the importance and the design of high-level declarative query languages [9, 12, 2, 5, 19, 17]. Most, if not all, of the second generation commercial OODB's provide or will provide some form of high-level declarative query languages [27, 14, 26, 20, 21, 23].

The query languages, like those of the relational model, transfer the burden of choosing an efficient execution plan for a query to the database system. This causes a resurrection of the study of query optimization problem in the object-oriented setting [15, 22, 28, 30, 8, 31, 7, 18]. Optimization of queries in an OODB is, for various reasons, inherently difficult [4, 15, 22]. Indeed, critics of the object-oriented approach frequently point to the theoretical limitations to query optimization as a major drawback of the object-oriented approach as compared with the relational approach

Although the optimization problem in general is inherently difficult for object-oriented queries, it would be highly desirable if techniques could be developed to assist the efficient processing of important subclasses of queries in such an environment. In an OODB, classes are named collections of similar objects. A class could be refined into subclasses. Subclasses are specialization of their superclasses. Specialization of a class is achieved by refining and/or adding properties to its superclasses. Since properties of a superclass are also properties of its subclasses, a

subclass is said to inherit the properties of its superclasses. Class-subclass relationships form an acyclic directed graph called inheritance or generalization hierarchy. Properties are attributes or methods defined on types; they are applicable only to instances of the types. In effect, therefore, types are constraints imposed on objects in the classes. A natural first step in query optimization is to use the typing constraints implied by the schema to minimize the search space for variables involved in the query. Properties are formally denoted as attribute-type pairs in this paper.

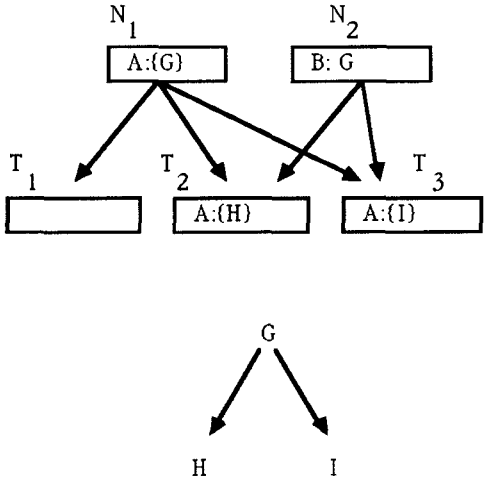
Example 1.1 *The following is a schema for a vehicle rental database.*



In this application, Auto, Trailer and Truck are subclasses of the superclass Vehicle. There is a subclass of clients, called discount customers, who receive a special rate and are not required to make a deposit on the vehicles rented. However, discount customers only allow to rent automobiles, and not other types of vehicles.

Suppose we want to find out all those vehicles currently rented to discount clients. Express in a calculus-like language, the query looks like $\{ x \mid \exists y\ (x \in Vehicle\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$. *Since discount clients are allowed to rent Auto only, the above query is equivalent to the following query:* $\{ x \mid \exists y\ (x \in Auto\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$. *The latter query is considered more optimal since the search space for each variable is minimal, given the typing constraints imposed by the schema.* □

Example 1.2 *Let us consider the following database schema and let us assume that objects in classes* N_1, N_2 *and* G *are partitioned by objects in their corresponding subclasses.*



Consider the following query:

Q: $\{ x \mid \exists y\ \exists s\ (x \in N_1\ \&\ y \in G\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)\}$.

The query Q *retrieves all those objects in* N_1 *such that the A-component of the object contains its B-component as well as an object from the class H. Since* x *ranges over the class* N_1 *and* N_1 *is partitioned by its subclasses,* x *is an object from* T_1, T_2 *or* T_3. *The object denoted by* x *cannot be an object from* T_1 *because* T_1 *does not have the attribute B. Then either* x *is an object in the class* T_2 *or an object in the class* T_3. *If* x *denotes an object from* T_3, *then its A-component contains objects from the class I. Consequently, the condition '*$s \in x.A$*' is unsatisfiable. Hence* x *can only range over objects in class* T_2. *In fact,* Q *is what we will call a positive conjunctive query. By a result in Section 4,* Q *is equivalent to the following union of queries:* $\{ x \mid \exists y\ (x \in T_2\ \&\ y \in H\ \&\ y = x.B\ \&\ y \in x.A)\} \cup \{ x \mid \exists y\ \exists s\ (x \in T_2\ \&\ y \in I\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)\}$. *The above union of queries is optimal in the sense that the variable search space is minimal among all the unions of positive conjunctive queries.* □

Inequalities could be implied by conditions in a query. This makes our problem even more complex.

Example 1.3 *Let* C, T_1 *and* T_2 *be distinct unrelated classes in a schema with* T_1 *and* T_2 *are subtypes or subclasses of the type of* $C.A$, *where* A *is an attribute of* C. *Consider the following queries:*

Q_1: $\{ x \mid \exists y\ \exists s\ \exists t\ (x \in C\ \&\ y \in C\ \&\ s \in T_1\ \&\ t \in T_2\ \&\ s = x.A\ \&\ t = y.A\ \&\ x \neq y)\}$.

Q_2: $\{ x \mid \exists y\ \exists s\ \exists t\ (x \in C\ \&\ y \in C\ \&\ s \in T_1\ \&\ t \in T_2\ \&\ s = x.A\ \&\ t = y.A)\}$.

Since T_1 and T_2 are distinct unrelated classes, the objects denoted by the variables s and t are distinct objects. This implies the inequality of x and y in both Q_1 and Q_2. By a result in Section 3, Q_1 and Q_2 are in fact equivalent. □

Most work on query optimization in the area of object-oriented databases concentrated on complex object optimization without considering the inheritance hierarchy [15, 22, 28, 30, 31, 7, 18]. Type checking of queries in the presence of non-strict inheritance hierarchy was studied in [8]. Our work is different from previous approaches in several important respect. Firstly, we use the typing constraints imposed by the inheritance hierarchy to study the containment and equivalence of queries. Secondly, our optimization is an exact minimization while the previous work are basically algebraic transformations and/or heuristic [15, 22, 28, 30, 31, 7]. Lastly, we derive algorithms for containment and minimization, and not just studying the decidability of equivalence as in [18]. However, in order to prove our results, we restrict ourselves to a special kind of queries which we will call positive conjunctive queries.

In the relational model, the class of conjunctive queries [11, 3] represents a natural and important subclass of relational queries that most often asked by a user. In this paper, we propose an object-oriented counterpart of conjunctive queries. The proposed conjunctive queries are essentially constructed from existential quantifiers, equalities, set memberships and negations. As was shown in Example 1.3 above, negations are inevitably implied by 'positive' conditions in a query in an object-oriented setting. As a matter of fact, all the queries in the above examples are instances of conjunctive queries. We then study the containment and minimization problems for subclasses of such queries.

The next section defines the class of conjunctive queries and some basic notation. In Section 3, we investigate the containment condition for terminal conjunctive queries. We characterize the containment, and hence equivalence, conditions for terminal conjunctive queries as well as for some of its interesting subclasses. In Section 4, we study the class of positive conjunctive queries and we find the containment and equivalence conditions. We also derive an algorithm for obtaining an exact minimization of positive conjunctive queries. The notion of minimization captures the intuition of minimization of the variable search space in a class of queries. Finally, we give our conclusions in Section 5.

2 Definitions and Notation

2.1 Classes and Schemas

In this section, we just briefly define some basic notation used throughout this discussion. Following [24], we introduce the notion of schema. A *schema* **S** is a triple ($\mathbf{C}$, σ, $\prec$), where $\mathbf{C}$ is a set of class names, σ is a function from $\mathbf{C}$ to tuple types, and $\prec$ is a partial order on $\mathbf{C}$. Let *type-expr*($\mathbf{C}$) be the set of all types that only involve class names in $\mathbf{C}$. The mapping σ associates a class name a tuple type in *type-expr*($\mathbf{C}$) which describes its structure. As noted in [16], there is no loss of representation power by restricting the structures of classes to tuple types. As in [24], we only consider schemas that are *consistent*.

We have class names in a schema. The $\prec$ relationship among classes represents the user-defined *inheritance* or *generalization hierarchy*. $A \prec B$ denotes the class A is a *subclass* of B. Subclasses are obtained by refining and/or adding properties to its superclasses. We assume the hierarchy has no cycle of length greater than 1. A class $A \in \mathbf{C}$ is said to be *terminal* if there is no other class $B \neq A$ such that $B \prec A$. Otherwise A is *non-terminal*. A class B is a *descendant* of a class A if $B \prec A$. Following [2, 24], we derive from this hierarchy a *subtyping* relation among expressions in *type-expr*($\mathbf{C}$). Throughout the discussion, we assume what we will call the *Terminal Class Partitioning Assumption*. That is, given any legal state, objects in every non-terminal class are partitioned by objects in its terminal descendants. This assumption was stated explicitly in Example 1.2.

2.2 A Class of Conjunctive Queries with Negation

In this section, we define a calculus-like query language for an object-oriented database. The language defined below modeled after query languages in systems like O_2 and Orion [27, 21], and is similar to a language proposed for a complex object model [1]. In this query language, users extract data from objects in a state by specifying a condition in the query. Query languages which allow explicit creation of object identifiers are also proposed and studied in the literature [2, 19, 17].

Queries are constructed from a set of variables, equality operators '$=$' and '$\neq$', membership operators '$\in$' and '$\notin$', logical operators '&' and '$\vee$', as well as existential quantifiers.

First we define the concept of term. Terms enable us to refer to a component in an object. Syntactically, a *term* $f(x)$ is of the following form: x or $x.A$, where

x is a variable and A an attribute.

An *atom* or an *atomic formula* is defined as one of the following:

1. $x\ \theta\ C_1 \vee \cdots \vee C_n$, where θ is one of $\{\in, \notin\}$, C_i's are class names, and x a variable. An atom $x \in C_1 \vee \cdots \vee C_n$ is called a *range* atom and it asserts that the variable x is an object in C_i, for some $1 \leq i \leq n$. An atom $x \notin C_1 \vee \cdots \vee C_n$ is called a *non-range* atom and it asserts that the object the variable x represents cannot be a member of class C_i, for any $1 \leq i \leq n$.

2. $g(x)\ \theta\ h(y)$, where $g(x)$ and $h(y)$ are terms involving variables x and y, respectively, θ is one of $\{=, \neq\}$. The atoms $g(x) = h(y)$ and $g(x) \neq h(y)$ are called *equality* and *inequality* atoms, respectively. The equality atom asserts that the operands denote the identical object. Likewise, an inequality atom asserts that the operands denote different objects.

3. $x\ \theta\ y.A$, where θ is one of $\{\in, \notin\}$. The atoms $x \in y.A$ and $x \notin y.A$ are called *membership* and *non-membership* atoms, respectively. A membership atom asserts that the object denoted by x is a member of the set object denoted by $y.A$. A non-membership atom asserts that x is not a member of $y.A$.

It is worth noting that *path expressions* of the form $x.A_1 \ldots A_n$, and atoms of the form $y.A\ \theta\ C_1 \vee \cdots \vee C_n$ or of the form $x.A\ \theta\ y.B$, where θ is one of $\{\in, \notin\}$, can be represented indirectly in our language. A *formula* is constructed from atomic formulas, logical operators '&' and '∨', as well as existential quantifiers. *Bound* and *free* variables are defined in the usual manner. A query is an expression of the form $\{\ s_0 \mid f(s_0, s_1, \ldots, s_m)\}$, where all occurrences of s_0 are free and s_i's, $1 \leq i \leq m$, are the set of bound variables in the formula f. A query $\{\ s_0 \mid f(s_0, s_1, \ldots, s_m)\}$ is *conjunctive* if $f(s_0, s_1, \ldots, s_m)$ is of the form $Q_1 s_1 \ldots Q_m s_m (M)$, where every $Q_p s_p$ is an existential quantifier, M is a formula containing no quantifier, and M is a conjunction of atomic formulas. $Q_1 s_1 \ldots Q_m s_m$ is called the *prefix* and M is called the *matrix* of the formula or the query. An atom is *positive* if it is a range, equality or membership atom. A conjunctive query $\{\ s_0 \mid f(s_0, s_1, \ldots, s_m)\}$ is *positive* if it involves only positive atoms.

A component of an object may have an unknown value. Consequently, we introduce the null value 'Λ' as a possible attribute value for an object. With null values are allowed, a logic, called *3-value* logic, is used to evaluate queries [13].

Given a state s, an answer to a query is defined in the usual way. That is, the free variable in the query is first mapped to an object via an assignment α. An *answer* of $\{\ s_0 \mid f(s_0, s_1, \ldots, s_m)\}$ w.r.t. s via α is $\alpha(s_0)$, if the closed formula $f(\alpha(s_0), s_1, \ldots, s_m)$ is evaluated to true in s using the 3-value logic. If Q is a query and s is a state, *the answer* of the query w.r.t. s, denote $Q(\mathbf{s})$, is the collection of answers of the query w.r.t. s. A query Q is *satisfiable* if there is a state s such that $Q(\mathbf{s})$ is non-empty. Given two queries F and G (on a schema $\mathbf{S}$), F is said to *contain* G, denotes $F \supseteq G$, if $F(\mathbf{s}) \supseteq G(\mathbf{s})$, for all states s on $\mathbf{S}$. Two queries F and G are said to be *equivalent*, denotes $F \equiv G$, if they contain each other.

2.3 Well-formed Conjunctive Queries

We consider only those queries in which each term either denotes an object, or a set of objects, but not both. We called such class of queries *well-formed* queries. Well-formed queries include *safe* as well as *unsafe* queries that produce infinite answers [33]. The following defines when a query is well-formed.

Given a conjunctive query, additional equalities among terms can be inferred with the following algorithm. The edges labelled with '=' in a graph in the following algorithm are called *equality* edges.

Algorithm EqualityGraph: Given a conjunctive query, generate additional implied equality edges.

Input: A conjunctive query Q.

Output: A graph $E(Q)$, called the *complete equality relationship graph* for Q.

Method:

(1) Generate a graph with terms in Q as nodes. Generate equality edges by applying the following three steps exhaustively to the graph until no more edges can be derived.

(i) For each term $f(x)$, derive the equality edge $f(x)=f(x)$. For each equality atom '$f(x)=g(y)$', generate an equality edge, between the node '$f(x)$' and the node '$g(y)$', if no such edge exists between them.

(ii) If $f(x)=g(y)$ and $g(y)=h(z)$ are equality edges, then derive the equality edge $f(x)=h(z)$, if no such edge exists between them.

(iii) If x and y are variable nodes, $x = y$ is an equality edge, and both $x.A$ and $y.A$ are nodes in the graph, then derive the equality edge $x.A = y.A$, if no such edge exists between them.

(2) Output the graph constructed.

Given the complete equality relationship graph $E(Q)$ for a conjunctive query Q, define an equivalence relation as follows. For each term $f(x)$ in $E(Q)$, define $[f(x)]$ to be $\{g(y) \mid g(y)$ is a node in $E(Q)$ and there is an equality edge between $f(x)$ and $g(y)\}$. Since equality is an equivalence relation, it partitions terms in $E(Q)$. The equivalence classes defined above are said to be *equivalence classes* in $E(Q)$.

Let Q be a query. An occurrence of a term $f(y)$ in the matrix of Q is a *set occurrence* if the occurrence appears on the right-hand side of a membership or non-membership atom. All other occurrences in the matrix of Q are *object occurrences*. A term $f(x)$ is an *object* term if there is an object occurrence of $g(y) \in [f(x)]$ in the query. A term $f(x)$ is an *set* term if there is a set occurrence of $g(y) \in [f(x)]$ in the query.

A conjunctive query Q is *well-formed* if (i) every term in Q is either an object term or a set term, but *not* both, (ii) each object term of the form $x.A$ is equated to some variable, that is, there is a variable z in the equivalence class $[x.A]$, and (iii) every variable in Q ranges over exactly one disjunction of classes, that is, there is exactly one range atom associated with each variable.

The conditons (ii) and (iii) are not really restrictions, since an object term can always be equated to some distinct variable and every variable can range over all classes in the schema. If there is a variable ranges over more than one disjunction, then by introducing new variables and equalities of variables, the original query can be converted into an equivalent query in which every variable ranges over exactly one disjunction. These two conditions are needed to simplify the discussion in the subsequent sections. Throughout this discussion, we use the term a conjunctive query to mean a well-formed conjunctive query.

2.4 Transforming Conjunctive Queries into Unions of Terminal Conjunctive Queries

A conjunctive query Q is *terminal* if range atoms in Q are of the form '$x \in C$', where C is a terminal class name. Given a conjunctive query Q, it can always be converted into an equivalent union of terminal conjunctive queries.

Example 2.1 *Consider the conjunctive query in Example 1.1:* $\{ x \mid \exists y\ (x \in Vehicle\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$. *Vehicle is not a terminal class while Discount is. Since there are three types of vehicles and assuming that all vehicle objects are partitioned by objects in these subclasses, then the query is equivalent to the following union of terminal conjunctive queries:*

$\{ x \mid \exists y\ (x \in Auto\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$
$\cup\{$
$x \mid \exists y\ (x \in Trailer\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$
$\cup\{\ x \mid \exists y\ (x \in Truck\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$. □

Proposition 2.1 *Let* $Q = \{ t \mid \exists \mathbf{s}\ h(\mathbf{s}, t)\}$ *be a conjunctive query. Then* Q *can always be converted into an equivalent union of queries of the following form:* $Q_1 = \{ t \mid \exists \mathbf{s}\ h_1(\mathbf{s}, t)\} \cup \cdots \cup Q_n = \{ t \mid \exists \mathbf{s}\ h_n(\mathbf{s}, t)\}$, *where each* Q_i *is terminal and conjunctive.*

[Proof]: Omitted. □

2.5 Testing Satisfiability of Terminal Conjunctive Queries

In this subsection, we turn to the problem of determining if a terminal conjunctive query is satisfiable.

Theorem 2.2 *There is an efficient algorithm for determining if a terminal conjunctive query is satisfiable.*

[Proof]: See [10]. □

Given a satisfiable terminal conjunctive query Q, the non-range atoms in Q can be removed without changing the answer of Q. We assume from now on that a satisfiable terminal conjunctive query has no non-range atom.

3 Containment of Terminal Conjunctive Queries

In this section, we study and characterize the containment condition for two terminal conjunctive queries in our language. We assume in this section that a given terminal conjunctive query is satisfiable. The containment condition is characterized by what we will call non-contradictory variable mappings.

3.1 Non-Contradictory Variable Mappings

Let $Q = \{t \mid h(\mathbf{s}, t)\}$ be a query and $S = \{A_1, \ldots, A_n\}$ a set of atoms defined on variables in Q. We denote $Q\&S$ as $\{t \mid h(\mathbf{s}, t)\ \&\ A_1\ \&\ A_2\ \&\cdots\&A_n\}$.

We now define the concept of *derivability of positive atoms* from a given query. Recall that an atom is

positive if it is of the form '$x \in C$', '$f(x) = g(y)$', or '$x \in g(y)$'. Let Q be a terminal conjunctive query and $E(Q)$ be the complete equality relationship graph for Q.

Q is said to *derive* $x \in C$, denoted as $Q \vdash x \in C$, if and only if '$x \in C$' is an atom in Q.

Q is said to *derive* $f(x) = g(y)$, denoted as $Q \vdash f(x) = g(y)$, if and only if there are $s \in [x]$ and $t \in [y]$ such that $f(s)$ and $g(t)$ are object terms in Q and $f(s) \in [g(t)]$.

Q is said to *derive* $x \in y.A$, denoted as $Q \vdash x \in y.A$, if and only if there is $s \in [x]$ and $t \in [y]$ such that '$s \in t.A$' is an atom in Q.

We define when a query contradicts an inequality or non-membership atom as follows. As pointed out in Section 2.5, we do not have to consider non-range atoms.

Q does not contradict $f(x) \neq g(y)$ if and only if there are $s \in [x]$, $t \in [y]$ such that $f(s)$ and $g(t)$ are both object terms in Q and $Q \& \{f(s) \neq g(t)\}$ is satisfiable.

Q does not contradict $x \notin y.A$ if and only if x is a variable in Q and there is $t \in [y]$ such that $t.A$ is a set term in Q and $Q \& \{x \notin t.A\}$ is satisfiable.

Let μ be a variable mapping from Q_2 to Q_1. The mapping μ from Q_2 to Q_1 is said to be *non-contradictory* if for every positive atom A in Q_2, $Q_1 \vdash \mu(A)$, and for every non-membership or inequality atom in Q_2, Q_1 does not contradict $\mu(A)$. Otherwise μ is *contradictory*.

Let Q be a terminal conjunctive query. A function τ on variables in Q is said to be a *standardization* function if for any pair x and y of variables in any equivalence class $[g(z)]$ in $E(Q)$, where $g(z)$ is an object term in Q, $\tau(x) = \tau(y) = m$, where m is a variable in $[g(z)]$.

3.2 Main Results

The containment condition is defined via non-contradictory mappings.

Example 3.1 *Let C and D be terminal classes and let $\{D\}$ be a subtype of type(C.B), where B is an attribute of C. Consider the following two terminal queries:*

Q_1: $\{ x \mid \exists y \exists z (x \in C \ \& \ y \in C \ \& \ z \in D \ \& \ z = y.A \ \& \ z \in y.B \ \& \ x = y)\}$.

Q_2: $\{ y \mid \exists z (y \in C \ \& \ z \in D \ \& \ z = y.A)\}$.

The query Q_2 retrieves objects in class C with a non-null A-component. There is a non-contradictory mapping μ from Q_2 to Q_1. The mapping μ is defined as follows: $\mu(y) = x$ and $\mu(z) = z$. We need to show that for each atom A in Q_2, $Q_1 \vdash \mu(A)$.

When A is $y \in C$, $\mu(A)$ is $x \in C$ which is clearly derivable from Q_1. Similarly when A is $z \in D$.

When A is $z = y.A$, $\mu(A)$ is $z = x.A$. Since $y \in [x]$ in $E(Q_1)$ and $y.A$ is an object term in Q_1, $z \in [y.A]$ in $E(Q_1)$. Therefore $Q_1 \vdash z = x.A$.

By a result in this section, $Q_1 \subseteq Q_2$. Informally, whenever there is a satisfying assignment α for Q_1, α maps $y.A$ (or $x.A$) to a non-null value. Hence $Q_1 \subseteq Q_2$.

The only variable mapping μ from Q_1 to Q_2 that preserves range atoms is to map x and y to the variable y and map z to z. However, $\mu(z) \in \mu(y.B) = z \in \mu(y).B = z \in y.B$ which is not derivable from Q_2. By a result in this section, $Q_2 \not\subseteq Q_1$. □

The presence of inequality and non-membership atoms makes our charcterization more complicated.

Example 3.2 *Let C be a terminal class. Consider the following three terminal conjunctive queries:*

Q_1: $\{ x \mid \exists y \exists z (x \in C \ \& \ y \in C \ \& \ z \in C \ \& \ x \neq y \ \& \ y \neq z)\}$.

Q_2: $\{ x \mid \exists y (x \in C \ \& \ y \in C \ \& \ x \neq y)\}$.

Q_3: $\{ x \mid \exists y \exists z (x \in C \ \& \ y \in C \ \& \ z \in C \ \& \ x \neq y \ \& \ y \neq z \ \& \ x \neq z)\}$.

At the first glance, perhaps the first two queries are not equivalent. It is easy to see that $Q_1 \subseteq Q_2$. However, $Q_2 \subseteq Q_1$ since the condition in Q_1 only requires the existence of two distinct objects to be satisfiable. Q_3 is not equivalent to Q_1 since the condition in Q_3 requires the existence of three distinct objects. Clearly $Q_3 \subseteq Q_1$. □

Example 3.3 *Let T_1 and T_2 be distinct terminal classes in a schema with T_1 as a subclass of type(T_2.A). Consider the following two terminal conjunctive queries:*

Q_1: $\{ x \mid \exists y (x \in T_1 \ \& \ y \in T_2)\}$.

Q_2: $\{ x \mid \exists y (x \in T_1 \ \& \ y \in T_2 \ \& \ x \notin y.A)\}$.

It is easy to see that $Q_2 \subseteq Q_1$. The reason that $Q_1 \not\subseteq Q_2$ is due to the fact that there is some state in which there is a satisfying mapping μ from Q_1 to object identifiers in the state with $\mu(x)$ as a member of $\mu(y.A)$, but there is no satisfying mapping α from Q_2 to s with $\alpha(x) = \mu(x)$. For states like these, an answer generated by Q_1 need not be generated by Q_2. □

Example 3.3 suggests that to determine if $Q_1 \subseteq Q_2$, we may have to augment Q_1 with some membership as well as equality atoms to represent all possible states from which the same answer is generated for Q_1.

Let Q be a query and S be a (possibly empty) set of equalities of variables from Q. Then $Q \& S$ is said to be *a consistent augmentation* of Q if $Q \& S$ is satisfiable.

We are now ready to state the containment condition for two terminal conjunctive queries.

Theorem 3.1 *Let $Q_1 = \{ t_1 \mid M_1\}$ and $Q_2 = \{ t_2 \mid M_2\}$ be terminal conjunctive queries. $Q_1 \subseteq Q_2$ if and only if for every consistent augmentation $Q_1 \& S$ of Q_1, and for every subset W of T, where $T = \{ x \in t.P \mid Q \& S \& \{x \in t.P\}$ is satisfiable, $t.P$ is a set term and x is a variable in $Q_1 \& S\}$, there is a variable mapping μ from Q_2 to $Q_1 \& S \& W$ satisfying the following:*

(i) $\tau(\mu(t_2)) = \tau(t_1)$, for any standardization function τ on $Q_1 \& S \& W$, and

(ii) μ is non-contradictory.

[Proof]: Omitted. □

An atom is said to be a *non-inequality* atom if it is *not* an inequality atom.

Corollary 3.2 *Let $Q_1 = \{ t_1 \mid M_1\}$ and $Q_2 = \{ t_2 \mid M_2\}$ be terminal conjunctive queries and that Q_2 involves only non-inequality atoms. Let $T = \{ x \in t.P \mid Q \& \{x \in t.P\}$ is satisfiable, $t.P$ is a set term and x is a variable in $Q_1\}$. $Q_1 \subseteq Q_2$ if and only if for every subset W of T, there is a variable mapping μ from Q_2 to $Q_1 \& W$ satisfying the following:*

(i) $\tau(\mu(t_2)) = \tau(t_1)$, for any standardization function τ on $Q_1 \& W$, and

(ii) μ is non-contradictory.

[Proof]: Omitted. □

Corollary 3.3 *Let $Q_1 = \{ t_1 \mid M_1\}$ and $Q_2 = \{ t_2 \mid M_2\}$ be terminal conjunctive queries and that Q_2 involves only positive and inequality atoms. $Q_1 \subseteq Q_2$ if and only if for every consistent augmentation $Q_1 \& S$, there is a variable mapping μ from Q_2 to $Q_1 \& S$ satisfying the following:*

(i) $\tau(\mu(t_2)) = \tau(t_1)$, for any standardization function τ on $Q_1 \& S$, and

(ii) μ is non-contradictory.

[Proof]: Omitted. □

The characterization of containment of positive terminal conjunctive queries is simpler.

Corollary 3.4 *Let $Q_1 = \{ t_1 \mid M_1\}$ and $Q_2 = \{ t_2 \mid M_2\}$ be terminal conjunctive queries and that Q_2 is positive. $Q_1 \subseteq Q_2$ if and only if there is a variable mapping μ from Q_2 to Q_1 satisfying the following:*

(i) $\tau(\mu(t_2)) = \tau(t_1)$, for any standardization function τ on Q_1, and

(ii) μ is non-contradictory.

[Proof]: Omitted. □

4 Containment and Minimization of Positive Conjunctive Queries

In this section, we first characterize the containment condition for positive conjunctive queries. We then derive an algorithm that, given a positive conjunctive query as the input, finds an optimal equivalent query among all the unions of positive conjunctive queries. The optimal query is expressed as a union of terminal positive conjunctive queries.

Theorem 4.1 *Let $M = Q_1 \cup \cdots \cup Q_s$ and $N = P_1 \cup \cdots \cup P_t$ be two unions of terminal positive conjunctive queries. $M \subseteq N$ if and only if for each Q_i in M, there is a P_j in N such that $Q_i \subseteq P_j$.*

[Proof]: Omitted. □

In the rest of this section, we derive an algorithm that finds an exact minimization for the class of positive conjunctive queries. The techniques used here are similar to those in [29], except that we have set membership operators and we do not use tableaux in the minimization process.

Let Q be a conjunctive query and x be a variable in Q. Define *term-class*$(Q, x) = \{E \mid x \in C_1 \vee \cdots \vee C_n$ is the range atom involving x in Q and E is a terminal descendant of some $C_i\}$. Informally, *term-class*(Q, x) gives the terminal descendent classes over which the variable x is ranging in the query.

We are now ready to define our notion of optimality. Let Q and P be two queries. Q is said to be *more optimal* than P, denote $Q < P$, if

1. Q and P are equivalent and
2. for each terminal class name C, the total number of occurrences of C in *term-class*(Q, y), for every variable y in Q, is less than or equal to the total number of occurrences of C in *term-class*(P, z), for every variable z in P.

A query Q is *search-space-optimal* among a set of queries **S** if for all P in **S** such that $P < Q$, $Q < P$. For search-space-optimal queries, the sets of objects logically accessed by the queries are minimal among all queries in **S**.

The following example illustrates the main idea behind our optimization technique.

Example 4.1 *Let us consider the database schema in Example 1.2. We argued that the following query is not minimal in the variable search space.*

Q: $\{ x \mid \exists y \exists s \ (x \in N_1 \ \& \ y \in G \ \& \ s \in H \ \& \ y = x.B \ \& \ y \in x.A \ \& \ s \in x.A)\}$.

Q is not a terminal query, by Proposition 2.1, Q is equivalent to the following union of terminal conjunctive queries.

Q_1:{ $x \mid \exists y\ \exists s\ (x \in T_1\ \&\ y \in H\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)$}.

Q_2:{ $x \mid \exists y\ \exists s\ (x \in T_2\ \&\ y \in H\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)$}.

Q_3:{ $x \mid \exists y\ \exists s\ (x \in T_3\ \&\ y \in H\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)$}.

Q_4:{ $x \mid \exists y\ \exists s\ (x \in T_1\ \&\ y \in I\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)$}.

Q_5:{ $x \mid \exists y\ \exists s\ (x \in T_2\ \&\ y \in I\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)$}.

Q_6:{ $x \mid \exists y\ \exists s\ (x \in T_3\ \&\ y \in I\ \&\ s \in H\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)$}.

Q_1 and Q_4 are unsatisfiable since B is not an attribute of T_1. Q_3 and Q_6 are unsatisfiable since $T_3.A$ is of type $\{I\}$ and therefore the condition 's∈x.A' is unsatisfiable. Hence Q is equivalent to $Q_2 \cup Q_5$. In fact, $Q_2 \cup Q_5$ is nonredundant since neither one contains the other. Q_2 can be minimized by mapping x to itself and by mapping y and s to y. The resulting minimized query is Q_2': { $x \mid \exists y\ (x \in T_2\ \&\ y \in H\ \&\ y = x.B\ \&\ y \in x.A)$}. By a result proven in this section, Q_2' and Q_5 cannot be minimized any more. Moreover, $Q_2' \cup Q_5$ is search-space-optimal among all the unions of positive conjunctive queries. □

A union of queries $Q_1(\mathbf{s}, t) \cup \cdots \cup Q_n(\mathbf{s}, t)$ is *nonredundant* if there are no Q_i and Q_j, $i \neq j$, such that $Q_i \subseteq Q_j$. Since we know the containment condition for terminal conjunctive queries, finding a nonredundant union for a union of terminal positive conjunctive queries can be done algorithmically. The following is an important property about nonredundant unions of terminal positive conjunctive queries.

Theorem 4.2 *Let $M = Q_1 \cup \cdots \cup Q_s$ and $N = P_1 \cup \cdots \cup P_t$ be two unions of nonredundant positive terminal conjunctive queries. $M \equiv N$ if and only if for each Q_i in M, there is a unique P_j in N such that $Q_i \equiv P_j$ and vice versa. Moreover, $s = t$.*

[Proof]: Omitted. □

Obtaining a nonredundant union is only the first step in finding a search-space-optimal query. The next step is to minimize the number of variables in each of the remaining subqueries. A *minimal* terminal conjunctive query of Q is an equivalent terminal conjunctive query with the minimal number of variables. We now show how to find minimal terminal positive conjunctive queries.

Let Q be a conjunctive query and μ be a variable mapping on Q. Then $\mu(Q)$ is a conjunctive query obtained by transforming the variables and atoms in Q with the mapping μ in the natural manner. A mapping μ is said to *preserve* a variable x if $\mu(x) \in [x]$.

Theorem 4.3 *Let Q be a terminal positive conjunctive query. Suppose there is a non-contradictory variable mapping from Q to itself that preserves the free variable. Then $\mu(Q)$ is equivalent to Q.*

[Proof]: Omitted. □

Corollary 4.4 *A terminal positive conjunctive query Q is minimal if for all non-contradictory variable mappings from Q to itself that preserve the free variable are bijective.*

[Proof]: Omitted. □

The following theorem describes an important property about minimal terminal positive conjunctive queries.

Theorem 4.5 *Let Q_1 and Q_2 be minimal terminal positive conjunctive queries. Suppose $Q_1 \equiv Q_2$. Then for all non-contradictory variable mappings from one query to the other, the mapping is a bijective function.*

[Proof]: Omitted. □

The resulting union of terminal positive conjunctive queries after removing redundant subqueries and minimizing the variables in the remaining subqueries is search-space-optimal among all the unions of positive conjunctive queries. The argument follows from Theorem 4.2 and Theorem 4.5.

5 Conclusion

Query optimization is an important and yet difficult problem in an OODB. The types of attributes in an inheritance hierarchy can be considered as constraints imposed on objects in a state. In this paper, we studied the containment and minimization problems for a class of natural queries called conjunctive queries. A conjunctive query can be expressed as an equivalent union of terminal conjunctive queries. We first characterized the containment, and hence equivalence, conditions for the class of terminal conjunctive queries. We then studied a subclass of conjunctive queries called positive conjunctive queries. We characterized the containment and equivalence conditions, as well as derived an algorithm for finding an exact minimization for the class of positive conjunctive queries. The equivalent minimized query is expressed as a union of terminal positive conjunctive queries with the property that the variable search space is minimal among all the unions of positive conjunctive queries. We shall investigate the minimization problem for conjunctive queries in general based on the result obtained in this paper.

References

[1] Abiteboul, S. and Beeri, C., "On the Power of Languages for the Manipulation of Complex Objects," Technical Report 846, INRIA, May 1988.

[2] Abiteboul, S. and Kanellakis, P., "Object Identity as a Query Language Primitive," *Proceedings of 1989 ACM SIGMOD*, Portland, Oregon, pp. 159-173. Also appears as Technical Report 1022, INRIA, April 1989.

[3] Aho, A.V., Sagiv, Y. and Ullman, J.D., "Equivalence of Relational Expressions," *SIAM J. of Computing 8(2)*, 1979, pp. 218-246.

[4] Bancilhon, F., "Object-Oriented Database Systems," *Proceedings of the 7th ACM Symposium on Principles of Database Systems*, Austin, Texas, 1988, pp. 152-162.

[5] Bancilhon, F., Cluet, S. and Delobel, C., "A Query Language for the O_2 Object-Oriented Database System," *Proceedings of the 2th International Workshop on Database Programming Languages* (Hull, R., Morrison, R. and Stemple, D., Eds.), Morgan Kaufmann Publishers Inc., San Mateo, Cal., 1990, pp. 122-138.

[6] Banerjee, J., Kim, W., and Kim, K.C., "Queries in Object-Oriented Databases," *Proceedings of the 4th International Conference on Data Engineering*, Feb. 1988, pp. 31-38.

[7] Beeri, C. and Kornatzky, Y, "Algebraic Optimization of Object-Oriented Query Languages," *Proceedings of the 3th International Conference on Database Theory*, Paris, France, Lecture Notes in Computer Science 470, December 1990, Springer-Verlag, pp. 72-88.

[8] Borgida, A. "Type Systems for Querying Class Hierarchies with Non Strict Inheritance," *Proceedings of the 8th ACM Symposium on Principles of Database Systems*, Philadelphia, Pennsylvania, 1989, pp. 394-401.

[9] Carey, M.J., DeWitt, D.J. and Vandenberg, S.L., "A Data Model and Query Language for EXODUS," *Proceedings of the ACM SIGMOD*, Chicago, 1988, pp. 413-423.

[10] Chan, E.P.F., "Testing Satisfiability of a Class of Object-Oriented Conjunctive Queries," Department of Computer Science, University of Waterloo, Technical Report, 1992.

[11] Chandra, A. and Merlin, P.M., "Optimal Implementation of Conjunctive Queries in Relational Databases," *Proceedings of the 9th ACM Symposium on Theory of Computing*, 1977, pp. 77-90.

[12] Cluet, S., Delobel, C., Lecluse, C. and Richard, P., "Reloop, an Algebra Based Query Language," *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases* (Kim. W., Nicolas, J-M and Nishio, S., Eds), Kyoto, Japan, December 1989, pp. 294-313.

[13] Codd, E.F., "Extending the Data Base Relational Model to Capture More Meaning," *ACM Transactions on Database Systems, 4(4)*, December 1979, pp. 397-434.

[14] Fisher, D.H., Beech, D., Cate, H.P., Chow, E.C., Connors, T., Davis, J.W., Derrette, N., Hoch, C.G., Kent, W., Lyngbaek, P., Mahbod, B., Neimat, M.A., Ryan, T.A., and Shan, M.C., "IRIS: An Object-Oriented Database Management System," *ACM Transactions on Office Information Systems, 1(5)*, January 1987, pp. 48-69.

[15] Graefe, G and Maier, D., "Query Optimization in Object-Oriented Database Systesm: A Prospectus," *Advances in Object-Oriented Database Systems* (K.R. Dittrich, Ed.), Lecture Notes in Computer Science 334, 1988, Springer-Verlag, pp. 358-362.

[16] Hull, R., "A Survey of Theoretical Research on Typed Complex Database Objects," *Databases* (J. Paredaens, Ed.), Academic Press, London, UK., 1987, pp. 193-256.

[17] Hull, R. and Yoshikawa, M., "ILOG: Declarative Creation and Manipulation of Object Identifiers (Extended Abstract)," *Proceedings of 16th International Conference on Very Large Data Bases*, Morgan Kaufmann, Los Altos, CA, 1990, pp. 455-468.

[18] Hull, R. and Yoshikawa, M., "On the Equivalence of Database Restructurings Involving Object Identifiers (Extended Abstract)," *Proceedings of the 10th ACM Symposium on Principles of Database Systems*, Denver, Colorado, 1991, pp. 328-340.

[19] Kifer, M., Lausen, G. and Wu, J., "Logical Foundations of Object-Oriented and Frame-based Languages," Technical Report 90/14 (revised), SUNY at Stony Brook, June 1990. To appear in *JACM*.

[20] Kim, W., "A Model of Queries for Object-Oriented Databases," *Proceedings of the 15th International Conference on Very Large Data Bases* (Apers & Wiederhold Eds), August 1989, Amsterdam, the Netherlands, Morgan Kauffman, CA.

[21] Kim, W., *Introduction to Object-Oriented Databases*, MIT Press, Cambridge, Massachusetts, 1990.

[22] Korth, H.F., "Optimization of Object-Retreival Query," *Advances in Object-Oriented Database Systems* (K.R. Dittrich, Ed.) Lecture Notes in Computer Science 334, 1988, Springer-Verlag, pp. 352-357.

[23] Lamb, C., Landis, G., Orenstein, J. and Weinreb, D., "The ObjectStore Database System," *CACM 34(10)*, October 1991, pp. 50-63.

[24] Lecluse, C., Richard, P., "Manipulation of Structured Values in Object-Oriented Database System," *Proceedings of the 2th International Workshop on Database Programming Languages* (Hull, R., Morrison, R. and Stemple, D., Eds.), Morgan Kaufmann Publishers Inc., San Mateo, Cal., 1990, pp. 113-121.

[25] Maier, D., Stein, J.C., Otis, A. and Purdy, A., "Development of an Object-Oriented DBMS," *Proceedings of OOPSLA '86*, Portland, Oregan, Sept. 1986, pp. 472-482, ACM, New York.

[26] *Ontos Object Database Programmer's Guide*, Ontologic, Inc., Burlington, MA, 1989.

[27] *The O_2 Programmer Manual*, O_2 Technology, 1991.

[28] Osborn, S.L., "Identity, Equality and Query Optimization," *Advances in Object-Oriented Database Systems* (K.R. Dittrich, Ed.), Lecture Notes in Computer Science 334, 1988, Springer-Verlag, pp. 346-351.

[29] Sagiv, Y. and Yannakakis, M., "Equivalence Among Relational Expressions with the Union and Difference Operators," *JACM 27(4)*, October 1980, pp. 633-655.

[30] Shaw, G. and Zdonik, S., "OO queries: Equivalence and Optimization," *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases*, (Kim. W., Nicolas, J-M and Nishio, S., Eds), Kyoto, Japan, December 1989, pp. 264-278.

[31] Shaw, G.M. and Zdonik, S.B., "An Object-Oriented Query Algebra," *Proceedings of the 2th International Workshop on Database Programming Languages*, (Hull, R., Morrison, R. and Stemple, D., Eds.), Morgan Kaufmann Publishers Inc., San Mateo, Cal., 1990, pp. 103-112.

[32] Ullman, J.D., "Database Theory-Past and Future," *Proceedings of the 6th ACM Symposium on Principles of Database Systems*, San Diego, CA, 1987, pp.1-10.

[33] Ullman, J.D., *Principles of Database and Knowledge-base Systems, Vol. I*, Computer Science Press, Rockville, Maryland, 1988.