Containment and Optimization of Object-Preserving Conjunctive Queries

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

Ron van der Meyden
School of Computing Sciences
University of Technology, Sydney
Australia,
ron@socs.uts.edu.au

March 12, 1999

Abstract

In the optimization of queries in an object-oriented database system (OODB), 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 study a class of queries for OODB's called *conjunctive queries*. Variables in a conjunctive query range over heterogeneous sets of objects. Consequently, a conjunctive query is equivalent to a union of conjunctive queries of a special kind, called *terminal* conjunctive queries. Testing containment is a necessary step in solving the equivalence and minimization problems. We first characterize the containment and minimization conditions for the class of terminal conjunctive queries. We then characterize containment for the class of all conjunctive queries, and derive an optimization algorithm for this class. The equivalent optimal query produced is expressed as a union of terminal conjunctive queries which has the property that the number of variables as well as their search spaces are minimal among all unions of terminal conjunctive queries. Finally, we investigate the complexity of the containment problem. We show that it is complete in Π_2^p.

1 Introduction

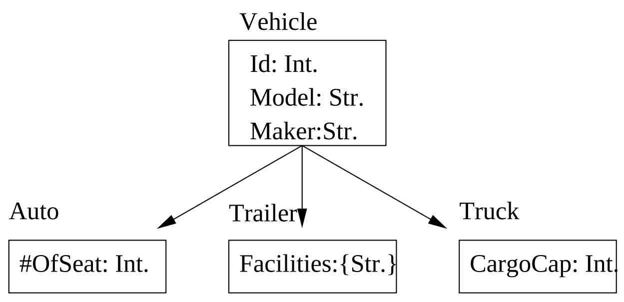
The initial attempts at constructing object-oriented databases (OODB's) provided only navigational programming languages for manipulating data [25, 6]. The lack of query languages like those available in relational systems has been criticized as a major drawback of the object-oriented approach [34, 5]. Consequently, most, if not all, commercial OODB's now provide, or will provide, some form of high-level declarative query language (e.g., [27, 15, 26, 21, 22]). These 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 has lead to a resurrection of the study of query optimization in the object-oriented setting (e.g., [30, 8, 7, 31, 18, 23, 13]). Most of these papers develop transformations that reduce the cost of evaluating a given query but do not necessarily produce an *optimal* equivalent query.

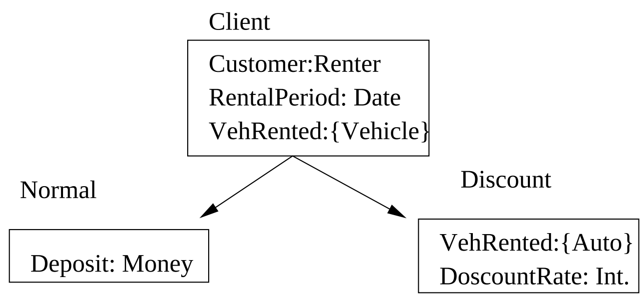
In the setting of relational databases, a well accepted notion of query optimality exists for the class of *conjunctive* queries [12], and the classical theory is based on the notion of query *containment*. A query Q_1 is said to be contained in a query Q_2 if in every database instance, the set of answers to Q_1 is a subset of the set of answers to Q_2. In this paper, we study the containment and optimization problems for a class of conjunctive queries in an object-oriented setting. The closely related *equivalence* problem has previously been addressed for object-oriented queries by Hull and Yoshikawa [18]. Our results are complementary to their work, in that the language in [18] is object-generating, while our language is *object-preserving*. Our language enables a user to retrieve objects from a database, but not to create new complex objects. Moreover, our language, like the one in [8], is defined on an inheritance hierarchy, whereas most languages studied in the literature are basically languages for complex objects without inheritance. The need to deal with inheritance introduces an extra level of complexity into the containment and optimization problems.

In an OODB, classes are named collections of similar objects. A class C may be refined into subclasses. Conversely, the class C is said to be a superclass of its subclasses. Subclasses are specializations of their superclasses. Consequently, objects in a class are also contained in its superclasses. Specialization of a class is often 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 an inheritance or generalization hierarchy.

Inheritance is a powerful modeling tool, because it allows for a better structured and more concise description of the schema, and helps in factoring out shared implementations in applications [4]. Objects belonging to the same class share some common properties. 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. Properties are formally denoted as attribute-type pairs in this paper. 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. The following example illustrates how this idea may be applied to the kind of object-oriented conjunctive query we consider.

Example 1.1 *The following is a schema for a vehicle rental database. It keeps track of all rental transactions for vehicles in the company. In this application, Auto, Trailer and Truck are subclasses of the superclass Vehicle. There are clients, called discount customers, who are known to the company and receive special treatment. Discount customers receive a special rate and are not required to pay a deposit on the vehicles rented. However, discount customers are only allowed to rent automobiles, and not other types of vehicles. Note that this constraint is captured by the more restrictive typing of the attribute VehRented in the subclass Discount of the class Client. Let us assume further that all superclasses are partitioned by their respective subclasses.*





Suppose we want to find all those vehicles that have been rented to a discount client. Expressed in a calculus-like language, the query looks like:

$$Q_1\text{: } \{\ x \mid \exists y\ \ (x{\in}\mathit{Vehicle}\ \&\ y{\in}\mathit{Discount}\ \&\ x{\in}y.\mathit{VehRented})\}.$$

Since discount clients are allowed to rent automobiles only, the above query is equivalent to the following query:

$$Q_2\text{: } \{\ x \mid \exists y\ (x{\in}\mathit{Auto}\ \&\ y{\in}\mathit{Discount}\ \&\ x{\in}y.\mathit{VehRented})\}.$$

Q_2 is considered to be more optimal since the number of variables as well as their search spaces are minimal, given the typing constraints implied by the schema. Let us consider another query. Assume that we want to find those clients who rented a truck. It can be expressed as follows:

$$Q_3\text{: } \{\ x \mid \exists y\ (x{\in}\mathit{Client}\ \&\ y{\in}\mathit{Truck}\ \&\ y{\in}x.\mathit{VehRented})\}.$$

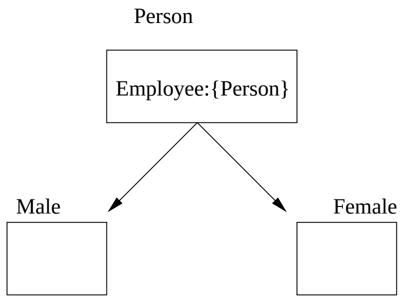
Since discount clients are allowed to rent automobiles only (but not other kind of vehicles), Q_3 is the same as the following query.

$$Q_4\text{: } \{\ x \mid \exists y\ (x{\in}\mathit{Normal}\ \&\ y{\in}\mathit{Truck}\ \&\ y{\in}x.\mathit{VehRented})\}.$$

□

Relational conjunctive queries have been studied extensively in the literature. A variable in a relational query ranges over a homogeneous relation. On the other hand, as illustrated by the example, variables in an object-oriented query range over classes which could consist of heterogeneous sets of objects. This is because a class may be refined to various subclasses, in which shared attribute names may correspond to different types or classes. For example, the variable x in Q_3 in Example 1.1 ranges over a heterogeneous set $Client = Normal \cup Discount$. All the members of this set have the attribute $VehRented$, but only for members x of $Normal$ can $x.VehRented$ contain an element of the class $Truck$. This implies clients who rent a truck are normal clients. This constitutes a significant divergence from the relational case. For instance, syntactically correct relational conjunctive queries are always satisfiable but this is not true for object-oriented conjunctive queries [10]. The additional complexity is also reflected in the containment problem, as is illustrated by the following example.

Example 1.2 *The following schema records the employer-employee relationships among a group of people. The Employee attribute indicates the set of employees hired by a person.*

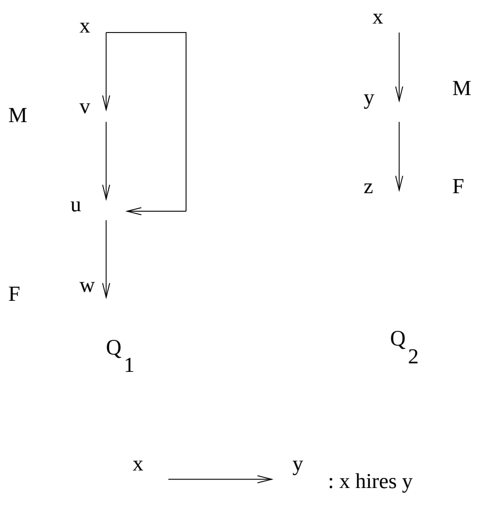


Consider the following two queries defined on the above inheritance hierarchy. Q_1 *retrieves all people* x *who hire a person* u *and a male* v *such that* u *is also an employee of* v *and* u *hires a female employee* w. Q_2 *finds all those people* x *who hire a male employee* y *who in turn hires a female employee* z. *Expressed in our language, they are as follows.*

$$Q_1\text{: } \{\ x \mid \exists u\ \exists v\ \exists w\ (x{\in}Person\ \&\ u{\in}Person\ \&\ v{\in}Male\ \&\ w{\in}Female\ \&\ u{\in}x.Employee\ \&\ v{\in}x.Employee\ \&\ u{\in}v.Employee\ \&\ w{\in}u.Employee)\}.$$

$$Q_2\text{: } \{\ x \mid \exists y\ \exists z\ (x{\in}Person\ \&\ y{\in}Male\ \&\ z{\in}Female\ \&\ y{\in}x.Employee\ \&\ z{\in}y.Employee)\}.$$

The above two queries are best visualized as the following two graphs.



We claim that Q_2 *contains* Q_1, *meaning that whenever there is an answer for* Q_1, *it will also be an answer for* Q_2. *The person* u *is either a male or a female. If* u *is a male, then* y *and*

z in Q_2 can be mapped to u and w, respectively. Similarly if u is a female, variables y and z in Q_2 can be mapped to v and u, respectively. Thus, whenever there is an answer for Q_1, it will also be an answer for Q_2. $\Box$

The above examples illustrate the kind of conjunctive queries we are interested in. Examples 1.1 and 1.2 demonstrate that the analysis of containment of conjunctive queries is more difficult than its counterpart in the relational case. This is due to the fact that the domains of attributes impose certain constraints on a query, and the analysis of the containment problem also involves analysis of disjunctive information.

The following is an overview of the problem and the approach we took in solving it. Given a conjunctive query $Q(\mathbf{S})$, where $\mathbf{S}$ is an object-oriented database schema denoted by an inheritance hierarchy, we want to find an equivalent query $Q'(\mathbf{S})$ that is, in some sense, optimal. Moreover, we are interested in determining when a conjunctive query is contained in another one. Both problems require an understanding of what a conjunctive query represents. We first observe that a conjunctive query, like the ones in Examples 1.1 and 1.2, can be decomposed as a union of special kind of conjunctive queries called terminal conjunctive queries. As typing constraints in an inheritance hierarchy are restrictions on objects in a database, not every terminal conjunctive query is satisfiable. With typing constraints implied by an inheritance hierarchy, unsatisfiable terminal conjunctive queries can be determined and are eliminated from a union. Having removed unsatisfiable terminal conjunctive queries, characterization of containment and optimization are then derived. The technique employed and the result obtained are similar to those in SPJU expressions in the relational theory [28].

Most work on query optimization in OODB's concentrates on complex object optimization without considering the typing constraints imposed by the inheritance hierarchy (e.g., [30, 7, 31, 18, 23, 13]). Type checking of queries in the presence of *non-strict* inheritance hierarchy was studied in [8]. Our work is different from all previous approaches in several important respects. Firstly, we use the typing constraints imposed by an inheritance hierarchy to study the containment, equivalence and optimization of queries. Secondly, our optimization is an exact minimization while most of the previous work deals with algebraic transformations and/or heuristics (e.g., [30, 31, 7, 23, 13]). Thirdly, with reasons similar to those noted in [28], characterizing equivalence does not suffice to solve the optimization problem. Instead, we need to understand the containment problem as well. This work, to our best knowledge, is the first work that provides a characterization for containment of queries in an object-oriented setting. This result could also find applications in view definition and classification in an OODB. For instance, to correctly integrate a virtual class or view into an inheritance hierarchy, it is imperative to resolve the containment problem for the view definition language [29]. Lastly, we demonstrate that the idea of containment mappings of relational conjunctive queries [12] can be extended to its object-oriented counterpart. Our proposed language, on the other hand, is perhaps more restrictive than some other query languages studied in the literature.

The next section defines the class of conjunctive queries and the basic notation needed throughout the discussion. In characterizing the containment and equivalence of terminal conjunctive queries, it is assumed that the query involved is satisfiable. We present an efficient algorithm for solving the satisfiability problem for terminal conjunctive queries in Section 3. The results in that section were proven in [10] and are needed in the subsequent discussions. Sections 4 and 5 characterize the containment, equivalence and minimization conditions for *terminal* conjunctive queries. In Section 6, we solve the containment problem and derive an algorithm for optimizing the class of all conjunctive queries. The notion of optimization captures the intuition of minimization of the number of variables as well as their search spaces. In Section 7, we analyse the complexity of testing containment of conjunctive queries. The main result shows that the problem is Π_2^p-complete. Finally, we give our conclusions in Section 8.

2 Definitions and Notation

In this section, we introduce notation that is necessary for the rest of the discussion.

2.1 Types, Classes and Schemas

We suppose given the following pairwise disjoint sets:

1. A set $\mathcal{T}$ of *atomic type domains*, where each atomic type domain is an *infinite* set of *atomic values*. Examples of atomic type domains are the set of integers, and the set of strings over some alphabet. We asssume distinct atomic type domains are disjoint.

2. A countably infinite set $\mathcal{O}$ of symbols which are called *object identifiers*.

3. A set $\mathcal{B}$ of *atomic types*, containing for each atomic type domain $T \in \mathcal{T}$, a symbol $\mathbf{T}$ naming that type domain. For brevity, we abuse notation by using the same symbol T for a type domain and its name. Thus $\mathcal{B}=\mathcal{T}$.

4. A countably infinite set $\mathcal{A}$ of symbols, the *attributes*.

5. A countably infinite set $\mathcal{C}$ of symbols which are called *classes*.

The set $\bigcup_{T\in\mathcal{T}} T$ is said to be the set of *atomic values*. The elements of $\mathcal{A}$ will be used as attribute names in tuple types, and the elements of $\mathcal{C}$ serve as names for user-defined classes.

A *type expression* over a set $\mathbf{C}\subseteq\mathcal{C}$ of class names is an expression defined as follows:

1. Every element of $\mathcal{B}$ is a type expression, called an *atomic type*.

2. Every element of $\mathbf{C}$ is a type expression, called a *class*.

3. If t is an atomic type or a class, then $\{t\}$ is type expression, called a *set* type.

4. If a_1 , ... , a_n are distinct attributes in $\mathcal{A}$ and t_1 , ... , t_n are atomic types, set types or classes, where $n\geq 0$, then $[a_1{:}t_1 , \ldots , a_n{:}t_n]$ is a type expression, called a *tuple* type. As in a relation scheme, the order of attributes is immaterial. The empty tuple [] is also a tuple type. The type t_i is said to be the *type* of the attribute a_i, for each $i = 1 \ldots n$.

We write *type-expr*($\mathbf{C}$) for the set of all type expressions over $\mathbf{C}$.

Following [24, 9], we introduce the notion of schema. A *schema* $\mathbf{S}$ is a triple ($\mathbf{C}$, σ, $\prec$), where $\mathbf{C}$ is a finite subset of $\mathcal{C}$, σ is a function from $\mathbf{C}$ to tuple types, and $\prec$ is a partial order on $\mathbf{C}$. The mapping σ associates to each class in $\mathbf{C}$ a tuple type in *type-expr*($\mathbf{C}$) which describes its structure. As noted in [16], there is no loss of representation power in restricting the structures of classes to be tuple types. The relationship $\prec$ among classes represents the user-defined *inheritance hierarchy.* We assume that 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 class $B \neq A$ such that $B \prec A$. Otherwise A is *non-terminal.* A class B is a *descendant* (or an *ancestor*) of a class A if $B \prec A$ (or $A \prec B$, respectively).

Following [2, 24], we derive from this hierarchy a subtyping relation $\leq$ among expressions in *type-expr*($\mathbf{C}$). Let $\mathbf{S} = (\mathbf{C}, \sigma, \prec)$ be a schema. The *subtyping* relation among expressions in *type-expr*($\mathbf{C}$) is the smallest ordering $\leq$ which satisfies the following axioms:

1. $A \leq A$ if $A \in \mathcal{B}$.
2. $B \leq C$ if $B \prec C$, for all classes B and C in $\mathbf{C}$.
3. $\{t\} \leq \{s\}$, for all types s, t such that $t \leq s$.
4. $[a_1{:}t_1 \ , \ldots , \ a_n{:}t_n \ , \ldots , \ a_{n+p}{:}t_{n+p}] \leq [a_1{:}s_1 \ , \ldots , \ a_n{:}s_n]$, for all atomic types, set types or classes $t_1 \ , \ldots , \ t_n$, $s_1 \ , \ldots , \ s_n$ such that $t_i \leq s_i$, for all $i = 1 \ldots n$.

The order of arguments in a tuple type is immaterial, so we have $[a_3 : C_3, a_2 : C_2, a_1 : C_1] \leq [a_1 : C_1, a_2 : C_2]$.

For any expressions E_1 and E_2 in *type-expr*($\mathbf{C}$), E_1 is a *subtype* of E_2 if $E_1 \leq E_2$. It is worth noting that the subtyping relation is a reflexive and transitive relation. As inheritance hierarchies are given by users, some schemas may not be meaningful. Let $\mathbf{S} = (\mathbf{C}, \sigma, \prec)$ be a schema. $\mathbf{S}$ is *consistent* if for all classes B and C such that $B \prec C$, we have $\sigma(B) \leq \sigma(C)$. We only consider consistent schemas throughout this paper. The schemas we have defined are essentially the same as those defined in ODMG-93 [9]. Thus our results are applicable to systems conforming to this standard. Let $C \in \mathbf{C}$. Attributes in $\sigma(C)$ are called the *attributes* of C. The *type of* $C.A$, denoted *type*($C.A$), is the type t of A in $\sigma(C)$. In consistent schemas, subclasses are specializations of their superclasses. Specialization of subclasses is represented formally by refining types of inherited attributes and/or adding new attribute-type pairs.

2.2 States, Domains and Objects

Let $\mathbf{S} = (\mathbf{C}, \sigma, \prec)$ be a schema and $\leq$ the subtyping relation on *type-expr*($\mathbf{C}$). Let $\mathbf{O}$ be a finite subset of $\mathcal{O}$ and $\mathbf{I}_c$ be a function from $\mathbf{O}$ to $\mathbf{C}$. Given $\mathbf{O}$ and $\mathbf{I}_c$, each type expression T in *type-expr*($\mathbf{C}$) is interpreted as a set of possible values, called the *domain* of T, denoted as $dom(T)$. In order to represent inapplicable attributes, we introduce a new symbol 'Λ'. The *domain* of a type with respect to (w.r.t.) $\mathbf{O}$ and $\mathbf{I}_c$ is defined as follows:

1. If $\mathbf{T} \in \mathcal{B}$ is an atomic type naming the type domain T, then $dom(\mathbf{T}) = T$.

2. For each class $D \in \mathbf{C}$, we define $dom(D) = \{o \mid o \in \mathbf{O}$ and $\mathbf{I}_c(o) = E$, where $E \prec D\}$.

3. For each set type $\{t\}$, we define $dom(\{t\}) = \{v \mid v \subseteq dom(t)\}$.

4. For each tuple type $[a_1{:}t_1 , \ldots , a_n{:}t_n]$, we define $dom([\ a_1{:}t_1 , \ldots , a_n{:}t_n]) = \{[a_1{:}v_1 , \ldots , a_n{:}v_n] \mid v_i \in dom(t_i) \cup \{\Lambda\}$ for all $i = 1 \ldots n\}$.

Note that the value of an attribute of a tuple may be the null value Λ. This is to be interpreted as the attribute being inapplicable.

A *state* $\mathbf{s}$ on a schema $\mathbf{S} = (\mathbf{C}, \sigma, \prec)$ is a triple $(\mathbf{O}, \mathbf{I}_c, \mathbf{I}_v)$, where $\mathbf{O}$ is a finite subset of $\mathcal{O}$, $\mathbf{I}_c$ is a function from $\mathbf{O}$ to $\mathbf{C}$, and $\mathbf{I}_v$ is a function from $\mathbf{O}$ to tuple values in domains of types with respect to $\mathbf{O}$ and $\mathbf{I}_c$. The function $\mathbf{I}_v$ maps each element in $\mathbf{O}$ to a tuple value which satisfies the following:

$$\forall o \in \mathbf{O}, \mathbf{I}_v(o) \in dom(\sigma(\mathbf{I}_c(o))).$$

That is, $\mathbf{I}_v$ defines the data value of an object and the (tuple) value of an object defined on a class must satisfy the type specification associated with the class. The set $\{<o, \mathbf{I}_v(o)> \mid o \in \mathbf{O}\}$ is the set of objects in the state $\mathbf{s}$. Two objects in a state are *identical* if and only if they have the same identifier, so we may sometimes abuse terminology by referring to the identifier o as an *object* of $\mathbf{s}$.

Let $[a_1{:}v_1 , \ldots , a_n{:}v_n]$ be a tuple value. Then $[a_1{:}v_1 , \ldots , a_n{:}v_n].a_i$ is v_i. We call v_i the value of attribute a_i in the tuple. If the attribute a does not occur in a tuple then the value of attribute a in the tuple is Λ.

In many, if not most, existing object-oriented database systems (e.g., [26, 21, 27]), an object is defined on exactly one most specialized class in an inheritance hierarchy. Consequently, as in [2, 21], we assume the following throughout the discussion.

Terminal Class Partitioning Assumption: Given any state $\mathbf{s}= (\mathbf{O}, \mathbf{I}_c, \mathbf{I}_v)$ on a schema $\mathbf{S}$, the class $\mathbf{I}_c(o)$ is a terminal class in $\mathbf{S}$ for every object o in $\mathbf{O}$. That is, every non-terminal class is partitioned by its terminal descendants.

2.3 A Class of Object-Preserving Conjunctive Queries

In this subsection, we define a calculus-like query language for an object-oriented database.

Queries are constructed from a set of variables, symbols from the set of atomic values, the equality operator '$=$', the membership operator '$\in$', the OR operator '$\sqcup$', the logical operator '&', as well as the existential quantifier '$\exists$'. The set of variables is assumed to be disjoint from other sets of symbols.

First we define the concept of term. Terms enable us to refer to an object or a component of an object. A *term* is an expression of one of the following forms: c or x or $x.A$, where c is an atomic value, i.e., c is in some atomic type domain, x is a variable and A is an attribute. A term of the form x or $x.A$ is called a *variable term.* An *attribute term* is of the form $x.A$.

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

1. $x\in C_1 \sqcup \cdots \sqcup C_n$, where the C_i's are classes or atomic types, and x is a variable. An atom $x\in C_1 \sqcup \cdots \sqcup C_n$ is called a *range* atom and it asserts that the variable x denotes an object in the class C_i or a value in the atomic type C_i, for some $1\leq i\leq n$.

2. $t_1 = t_2$, where t_1 and t_2 are terms. Such an atom is called an *equality* atom. An equality atom asserts that the operands denote identical objects or are of the same atomic value.

3. $x\in y.A$, where x and y are variables. The atom $x\in y.A$ is called a *membership* atom. A membership atom asserts that the object or atomic value denoted by x is a member of the *set object* denoted by $y.A$.

It is worth noting that path expressions of the form $x.A_1 \ldots A_n$ (as used in [37]) and of the form $x.A_1[y_1]. \ . \ . A_n[y_n]$ (as used in [19]); where x and y_i's are variables or atomic values, can all be represented indirectly in our language. Likewise, atoms of the forms $c{\in}x.A$ and $y.A{\in}C_1 \sqcup \cdots \sqcup C_n$ and of the form $x.A{\in}y.B$, where x and y are variables and c is an atomic value, can again be expressed indirectly in our language.

A *formula* is constructed from atomic formulas, the logical operator '&', as well as existential quantifiers. *Bound* and *free* variables are defined in the usual manner. A *query* is an expression of the form $\{\, t \mid \Phi(t)\}$, where t is either a variable or an atomic value and $\Phi(t)$ is a formula. The term t is called the *distinguished term* of the query. A query $\{\, s_0 \mid \Phi(t)\}$ is called *conjunctive* if $\Phi(t)$ is of the form $\exists s_1 ... \exists s_m(M)$, where M is a formula containing no quantifier that is a conjunction of atomic formulas.[1] $\exists s_1 ... \exists s_m$ is called the *prefix* and M is called the *matrix* of the formula or of the query. We also make use of *union queries*, which are expressions of the form $Q_1 \cup \ldots \cup Q_n$, where each Q_i is a conjunctive query.

2.4 Semantics of Queries

We now give the semantics of queries, and define the notion of query containment. It is convenient for technical reasons that will become apparent below to state the semantics in terms of a mapping to a language that uses the objects and atomic values of a state as basic syntactic entities. We remark that the semantics is slightly non-standard in that it requires that the terms occurring in an atom must have a non-null value in order for the atom to be true: this was handled in [10] using a three valued logic, but since only the truth of atoms is relevant to the containment question for conjunctive queries we simplify this here.

Define a *term over a state* $\mathbf{s} = (\mathbf{O}, \mathbf{I}_c, \mathbf{I}_v)$ to be an expression of one of the following forms: an atomic value c, an object $o \in \mathbf{O}$, or an expression $o.A$, where $o \in \mathbf{O}$ is an object and A is an attribute. The *value* $Val(t)$ of a term t over $\mathbf{s}$ is defined as follows.

1. If t is an atomic value c then $Val(t) = c$.

2. If t is an object $o \in \mathbf{O}$ then $Val(t) = o$.

3. If t is the expression $o.A$, where $o \in \mathbf{O}$ is an object and A is an attribute then $Val(t)$ is the value of attribute A in $\mathbf{I}_v(o)$.

[1] The results in this paper can be extended to conjunctive queries with more than one free variable.

Note that $Val(t)$ may be an atomic value, object, set, or the null value Λ.

An *atom over* $\mathbf{s}$ is an expression of one of the following forms:

1. $t \in C_1 \sqcup \cdots \sqcup C_n$, where t is a term over $\mathbf{s}$ and the C_i are classes or atomic types;

2. $t_1 = t_2$, where t_1 and t_2 are terms over $\mathbf{s}$, or

3. $t \in o.A$ where t is a term over $\mathbf{s}$, where o is an object of $\mathbf{s}$, and where A is an attribute.

We define certain atoms $\mathbf{A}$ over a state $\mathbf{s}$, to be *satisfied* in $\mathbf{s}$, written $\mathbf{s} \models \mathbf{A}$, as follows:

1. $\mathbf{s} \models t \in C_1 \sqcup \cdots \sqcup C_n$, where t is a term over $\mathbf{s}$ and each C_i is a class or atomic type, if $Val(t) \in dom(C_i)$ for some $i = 1 \ldots n$;

2. $\mathbf{s} \models t_1 = t_2$, where t_1 and t_2 are terms over $\mathbf{s}$, if $Val(t_1) = Val(t_2)$ and neither value is equal to Λ;

3. $\mathbf{s} \models t \in o.A$ where t is a term over $\mathbf{s}$, the o is an object of $\mathbf{s}$, and A is an attribute, if $Val(t)$ is not equal to Λ and $Val(o.A)$ is a set that contains $Val(t)$.

Note that in order for an atom to be satisfied, all of its terms must have non-null values.

An *assignment* for a query Q in a state $\mathbf{s} = (\mathbf{O}, \mathbf{I}_c, \mathbf{I}_v)$ is a function α mapping each variable of Q either to an atomic value or to an object in $\mathbf{O}$. Assignments may be extended to mappings from the terms and atoms of Q to terms and atoms over $\mathbf{s}$, respectively, as follows:

1. For terms which are atomic values c we define $\alpha(c) = c$.

2. For terms of the form $x.A$, where x is a variable, we define $\alpha(x.A)$ to be the expression $o.A$, where $o = \alpha(x)$.

3. For atoms $\mathbf{A}$ of Q we define $\alpha(\mathbf{A})$ to be the atom over $\mathbf{s}$ obtained by substituting for each term t in $\mathbf{A}$ the term $\alpha(t)$.

Using the notion of satisfaction of atoms over a state in that state, we now define a formula Φ to be satisfied in a state $\mathbf{s}$ with respect to an assignment α, written $\mathbf{s}, \alpha \models \Phi$, in the usual way. For atomic formulae $\mathbf{A}$ we have $\mathbf{s}, \alpha \models \mathbf{A}$ if $\mathbf{s} \models \alpha(\mathbf{A})$. The cases of Boolean operators and quantifiers are as in the standard semantics of first order logic, where the universe consists of the union of the sets $dom(T)$, where T ranges over all type expressions.

A query $Q = \{\ t \mid \Phi(t)\}$ is said to be *satisfied in a state* $\mathbf{s}$ *with respect to an assignment* α, written $\mathbf{s}, \alpha \models Q$, if $\mathbf{s}, \alpha \models \Phi(t)$. The assignment α is called a *satisfying* assignment for Q in this case. We say that the object or value a is an *answer* of Q with respect to $\mathbf{s}$ if there exists a satisfying assignment α for Q such that $a = \alpha(t)$, where t is the distinguished term of Q. If Q is a query and $\mathbf{s}$ is a state, we write $Q(\mathbf{s})$ for the set of all answers of Q with respect to $\mathbf{s}$. For union queries Q of the form $Q_1 \cup \ldots \cup Q_n$ we define $Q(\mathbf{s})$ to be the set $Q_1(\mathbf{s}) \cup \ldots \cup Q_n(\mathbf{s})$.

A query Q is said to be *satisfiable* if there is a state $\mathbf{s}$ such that $Q(\mathbf{s})$ is non-empty. Given two queries Q_1 and Q_2 (on a schema $\mathbf{S}$), Q_1 is said to *contain* Q_2 with respect to $\mathbf{S}$, denoted $Q_1 \supseteq Q_2$, if $Q_1(\mathbf{s}) \supseteq Q_2(\mathbf{s})$, for all states $\mathbf{s}$ on $\mathbf{S}$. Two queries Q_1 and Q_2 are said to be *equivalent* with respect to schema $\mathbf{S}$, denoted $Q_1 \equiv Q_2$, if they contain each other with respect to $\mathbf{S}$.

We note that for conjunctive queries, the existential quantifiers are not strictly essential: the query $\{t \mid \exists x_1 \ldots \exists x_n(\Phi)\}$ is equivalent to the query $\{t \mid \Phi\}$. Consequently, we assume henceforth for purposes of analysis that queries do not contain existential quatifiers. This yields the following simple characterization of satisfaction: α is a satisfying assignment for a conjunctive query Q in a state $\mathbf{s}$ if and only if $\mathbf{s} \models \alpha(\mathbf{A})$ for all atoms $\mathbf{A}$ of Q. (It is still sometimes convenient to write formulae with quantifiers in order to scope variables and avoid naming conflicts.)

2.5 Well-formed Conjunctive Queries

We consider only those queries in which each term either denotes an object or a value, or a set of objects or values, but not both. We call such queries *well-formed.* The following defines when a query is well-formed. First we note that, given a conjunctive query, additional equalities among terms could be inferred with the following algorithm. It is easy to see that the inferences performed in the algorithm are correct.

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

Input: A conjunctive query Q.

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

Method:

The edges $\{x, y\}$ in the graph *E(Q)* are called *equality* edges, and are also denoted by '$x = y$'.

(1) Generate a graph with terms in Q as nodes. (If $x.A$ is a term of Q then so is x.) Generate additional nodes and equality edges by applying the following three steps exhaustively to the graph until no more edges can be derived.

(i) For each node t, derive the equality edge $t = t$. For each equality atom '$s = t$' of Q, generate an equality edge between the node s and the node t.

(ii) If $s = t$ and $t = u$ are equality edges, then derive the equality edge $s = u$.

(iii) If x and y are variable nodes, $x = y$ is an equality edge, and $x.A$ is a node in the graph, then add the node $y.A$, if it does not already exist, and derive the equality edge $x.A = y.A$.

(2) Output the graph constructed.

By steps (i) and (ii), the complete equality relationship graph $E(Q)$ for a conjunctive query Q, yields an equivalence relation R, defined by tRt' if there exists an equality edge between t and t'. For each term t in $E(Q)$, the equivalence class $[t]$ of R containing t is the set $\{t' \mid t'$ is a node in $E(Q)$ and there is an equality edge between t and $t'\}$. These sets are said to be the *equivalence classes* of $E(Q)$.

Let Q be a query. An occurrence of a term $x.A$ in the matrix of Q is a *set occurrence* if the occurrence appears on the right-hand side of a membership atom. All other occurrences of terms in the matrix of Q are *object occurrences*. A term s is an *object* term if some term t in the equivalence class $[s]$ has an object occurrence in the query. A term s is a *set* term if there is a set occurrence in the query of some term $t \in [s]$. Intuitively, a set term is one that *must* denote a set, and an object term is one that *must* denote an object or atomic value. 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, and

(ii) each object term of the form $x.A$ is equated, directly or indirectly, to some variable or atomic value; that is, there is a variable or an atomic value in the equivalence class $[x.A]$, and

(iii) every variable in Q ranges over exactly one disjunction of classes or atomic types; that is, there is exactly one range atom associated with each variable.

Condition (i) is necessary for the satisfiability of the query, and arises from the obvious constraint that no term can simultaneously denote both an object and a set. It is worth remarking that this condition implies that a set term cannot occur within an equality atom in the query. For, such an occurrence would be an object occurrence, which would make the term simultaneously a set term and and object term. Note, moreover, that no object term can ever denote a set. For, by conditions (ii) and (iii), an object term must denote an element of some union of classes and atomic types.

For terms denoting objects or atomic values, condition (ii) is not a real restriction, since such a term can always be equated to some new existentially quantified variable ranging over all classes and atomic types. This condition is needed to simplify the discussion in the subsequent sections. In the case of condition (iii), note that because of the Terminal Class Partitioning Assumption, a query is unsatisfiable if it contains both $x \in C$ and $x \in D$, where C and D are distinct terminal classes. Such a query may be satisfiable if C and D are nonterminal classes, but in this case the two range atoms can be replaced (given a schema) with the single atom $x \in C_1 \sqcup \cdots \sqcup C_n$, where $C_1, \ldots, C_n$ are the common terminal descendants of C and D. Moreover, if the variable x occurs in no range atom, then we may clearly add the atom $x \in C_1 \sqcup \cdots \sqcup C_n$, where $C_1, \ldots, C_n$ are all terminal classes and atomic types, without changing the meaning of the query.

For the rest of this paper, we use the term *conjunctive queries* to denote well-formed conjunctive queries. Well-formed queries include *safe*, as well as *unsafe* queries that produce infinite answers [35].

2.6 Terminal Conjunctive Queries

A *terminal* conjunctive query is a conjunctive query in which every range atom is of the form '$x \in C$', where C is a terminal class or an atomic type.

Every conjunctive query can be expressed as a union of terminal conjunctive queries, as follows. First, define an *expansion* of a query Q to be a query obtained by replacing each range atom of the form $x \in C_1 \sqcup \cdots \sqcup C_n$ by one of the atoms $x \in C_i$. For example, the query

$$\{x \mid (x \in C \sqcup D) \wedge (y \in E \sqcup F) \wedge (x.A = y)\}$$

has four expansions, one of which is the query

$$\{x \mid x \in D \wedge y \in E \wedge x.A = y\}.$$

Note that every expansion of a conjunctive query is a terminal conjunctive query.

Proposition 2.1 *Let Q be a conjunctive query and let $Q_1, \ldots, Q_n$ be all the expansions of Q. Then Q is equivalent to $Q_1 \cup \ldots \cup Q_n$.*

[Proof]: See [10]. □

This result states that, semantically, a conjunctive query corresponds to a union of terminal conjunctive queries. Each terminal conjunctive query in a union could have variables defined on different domains. To solve the containment and equivalence problems, it is necessary to solve the satisfiability problem and to identify exactly the set of objects or values over which a variable is ranging. In Section 3, we present an algorithm for solving these problems for the terminal conjunctive queries. This algorithm employs typing constraints to determine satisfiability of a terminal conjunctive query. Queries that are not terminal could, by Proposition 2.1, first be decomposed into a union of terminal conjunctive queries. We can then apply the algorithm in Section 3 to each subquery in the union to determine its satisfiability, and delete the unsatisfiable subqueries from the union. In Sections 4 and 5, we shall derive algorithms for testing containment and for minimizing terminal conjunctive queries. Section 6.1 deals with containment of unions of conjunctive queries, which can be used to determine containment of arbitrary conjunctive queries.

3 An Efficient Algorithm for Testing Satisfiability of Terminal Conjunctive Queries

Testing satisfiability of restricted classes of conjunctive queries is an NP-complete problem [10]. However, determining if a terminal conjunctive query is satisfiable is tractable. We present in this section an algorithm that solves this problem in polynomial time, from [10], along with a sketch of its correctness proof. The aspect of this proof that is germane to our purposes in the present paper is that if the input query is satisfiable, it is possible to construct a 'minimal' state with respect to which the query returns a non-empty result. Various properties of the state constructed are needed in the proof of the characterization of containment of terminal conjunctive queries.

One of the reasons for unsatisfiability of a query is the incompatibility of the typing of its terms implied by the schema. Note that a query Q and schema $\mathbf{S} = (\mathbf{C}, \sigma, \prec)$ together determine a type *type(t)* for every term t of the query. (In every case, this type is in fact a class or atomic type.) For every atomic value c in an atomic type T, we define $type(c) = T$. For every variable x in Q, define $type(x)$ to be the unique class or atomic type C such that Q contains an atom of the form $x \in C$. For every term of the form $x.A$ in Q, define $type(x.A)$ as the type of attribute A in $\sigma(type(x))$, if $type(x)$ is a class and A is an attribute of $\sigma(type(x))$, and undefined otherwise.

In order for the query to be satisfiable, the type assigned to its terms must be consistent. By the Terminal Class Partitioning Assumption, terms denoting the same object must belong to the same terminal class or atomic type, and an object belonging to a set must be of a type admissible for that set. The following definition helps to check these conditions. If T is an atomic type or a class, we say D is a *terminal subtype* of T if either T is an atomic type and D is T, or D is a terminal descendant of T. Let t be an object term in Q. Define $SatType(t)$ to be the set of all D such that

1. for all terms $u \in [t]$, D is a terminal subtype of $type(u)$, and

2. for every '$u \in z.A$' in Q, where $u \in [t]$, D is a terminal subtype of C, where $type(z.A)$ is $\{C\}$.

Intuitively, $SatType(t)$ is the set of terminal types that are consistent with all the typing information on t derivable from the query. Since every object term must be equated to some variable, which must range over a terminal type, or to an atomic value, $SatType(t)$ contains at most a single element. Note that the definition depends only on the equivalence class of t, so we may also write $SatType([t])$ for $SatType(t)$. Note also that if Q is satisfiable, then $SatType(t)$

is a singleton set, for every variable object term t in Q. For, if $SatType(t)$ were empty, then it would be impossible to construct a satisfying assignment.

Algorithm SatTestUT: Verify if a terminal conjunctive query Q is satisfiable.

Input: A terminal conjunctive query Q on $\mathbf{S}$.

Output: *yes* if Q is satisfiable, and *no* otherwise.

Method:

Compute the complete equality relationship graph $E(Q)$ for Q.

(1) If there is an object term of the form $x.A$ for which $type(x.A)$ is undefined or equal to a set type, or there is a set term of the form $x.A$, for which $type(x.A)$ is undefined or is not equal to a set type, then output *no* and exit.

(2) If there is an object term t in Q such that $SatType(t)$ is empty, then output *no* and exit.

(3) If there is an object term t with two distinct atomic values c_1 and c_2 both in $[t]$, then output *no* and exit.

(4) Output *yes*.

Lemma 3.1 *If the algorithm SatTestUT outputs yes, then Q is satisfiable.*

Suppose the algorithm outputs *yes*. We construct a state $\mathbf{s}_Q$ and a satisfying assignment α for Q in $\mathbf{s}_Q$. This assignment will be called the *canonical* asssignment for Q in $\mathbf{s}_Q$. The details of the construction will be applied in later results.

First, to each equivalence class $[t]$ of the complete equality relationship graph of Q, we associate a distinct value $[t]_\alpha$ and a terminal class or an atomic type $type_\alpha([t])$. The type $type_\alpha([t])$ is defined to be $type(s)$ for any term s of the query in $[t]$. This is well-defined in the case of object terms by statement (2). For set terms $x.A$, note that if $t \in [x.A]$ then t must be of the form $y.A$ for some variable $y \in [x]$, by the fact that set terms do not occur in equations in Q and construction of $E(Q)$. Thus $type_\alpha([x.A])$ may be defined to be $type(x.A)$ in this case. The values $[t]_\alpha$ are assigned as follows.

Val1: If $type_\alpha([t])$ is an atomic type, and there is an atomic value c in $[t]$, then $[t]_\alpha = c$. By statement (3), c is the unique atomic value in $[t]$.

Val2: If $type_\alpha([t])$ is an atomic type and there is no atomic value in $[t]$, then take $[t]_\alpha$ to be any value of type $type_\alpha([t])$, in such a way that no two distinct equivalence classes are assigned the same value. (This is possible because the atomic types are infinite.)

Val3: If $type_\alpha([t])$ is a terminal class then the value $[t]_\alpha$ is defined to be just the equivalence class $[t]$ itself. In this case these values will be interpreted below as objects in the state to be constructed.

Val4: If none of the above cases apply, then $type_\alpha([t])$ is a set type. In this case $[t]_\alpha$ is defined to be the set $\{[v]_\alpha \mid$ '$v \in z.A$' is an atom in Q for some variable z in $[t]\}$. (Note that the variables v must be object variables, so the values $[v]_\alpha$ are already defined by cases Val1-Val3.)

Observe that it follows from this definition that distinct equivalence classes are assigned distinct values.

We now construct a state $\mathbf{s}_Q = (\mathbf{O}, \mathbf{I}_c, \mathbf{I}_v)$. We take the set of objects $\mathbf{O}$ of this state to be $\{ [t] \mid [t]$ is an equivalence class of Q such that $type_\alpha([t])$ is a terminal class$\}$. These objects are assigned to terminal classes by putting $\mathbf{I}_c([t]) = type_\alpha([t])$, for every $[t] \in \mathbf{O}$. The mapping $\mathbf{I}_v$, defined below, maps each $[t] \in \mathbf{O}$ to a tuple value on $type_\alpha([t])$. First, for each $[t] \in \mathbf{O}$, define $attr([t])$ to be the set of attributes A such that $x.A$ is a term of Q for some variable $x \in [t]$. We now define attribute values for the object $[t]$ as follows. Note that by conditions (1) and (2), the set $attr([t])$ is a subset of the set of attributes in $\mathbf{I}_c([t])$. For each attribute $A \in attr([t])$, the value assigned to the attribute A for the object $[t]$ is $[x.A]_\alpha$, where x is any variable in $[t]$ such that $x.A$ is a term of Q. (Note that the definition of the complete equality relationship graph ensures that if x and y are variables in $[t]$ such that $x.A$ and $y.A$ are terms of Q then $[x.A] = [y.A]$, so this definition is independent of the choice of x.) If A is an attribute in $type_\alpha([t])$ but not in $attr([t])$, a null value is assigned to A for the object $[t]$. This completes the definition of the state $\mathbf{s}$.

Finally, we define a mapping α from the variables of Q to this state. For each variable x, we let $\alpha(x)$ be the value $[x]_\alpha$.

Lemma 3.2 *The query Q is satisfied in the state $\mathbf{s}_Q$ under the assignment α.*

See [10] for more details (on a slightly different approach to the proof).

Theorem 3.3 *A terminal conjunctive query Q on $\mathbf{S}$ is satisfiable if and only if the algorithm SatTestUT outputs yes.*

[Proof]: See [10]. □

Unless otherwise stated, we consider only satisfiable terminal conjunctive queries for the rest of this paper.

4 Containment of Terminal Conjunctive Queries

We now set about developing a condition that characterizes containment of terminal conjunctive queries. We remark that some of the results of this section depend crucially on the notion of atoms over a state defined in section 2.4. For the rest of Sections 4 and 5, a query Q refers to a terminal conjunctive query Q.

We begin by defining a relation that is intended to capture the equations that must be satisfied under any satisfying assignment for a query. Recall that the terms in an equation must have non-null interpretations for the equation to hold. Given a query Q, define the relation $\approx$ on the set of terms by $s \approx t$ if either

1. s and t are both terms in the complete equality relationship graph of Q and $[s] = [t]$, or
2. s and t are the same term, which is an atomic value.

Note that the relation $\approx$ is an equivalence relation when restricted to the set of terms of the complete equality relationship graph of Q. However, $\approx$ is not an equivalence relation in general, since we do not have $t \approx t$ for terms t that are not an atomic value or in the complete equality relationship graph of Q. Intuitively, this reflects the fact that such terms may have interpretation Λ under a satisfying assignment, so that the equation $t = t$ does not hold. (We do have $c \approx c$ for atomic values c because the equation $c = c$ will hold under every satisfying assignment.)

We can extend the relation $\approx$ on terms to a relation on atoms by defining $\mathbf{A} \approx \mathbf{A}'$ if $\mathbf{A}$ and $\mathbf{A}'$ are atoms of the same syntactic form and the terms in corresponding positions are $\approx$-related. (For example, if $[x] = [y]$ and $z.A$ is a term of the complete equality relationship graph then we have $x \in z.A \approx y \in z.Z$, but not $x = z.A \approx y \in z.Z$, because these atoms are of different syntactic forms.)

Lemma 4.1 *If α is a satisfying assignment for Q in a state $\mathbf{s}$ then*

(i) If s and t are terms with $s \approx t$ then both $Val(\alpha(s))$ and $Val(\alpha(t))$ are non-null and $Val(\alpha(s)) = Val(\alpha(t))$.

(ii) If $\mathbf{A}$ and $\mathbf{A}'$ are atoms with $\mathbf{A} \approx \mathbf{A}'$ then $\mathbf{s} \models \alpha(\mathbf{A})$ if and only if $\mathbf{s} \models \alpha(\mathbf{A}')$.

[Proof]: The claim of part (i) is trivial for constants c, since we always have $Val(\alpha(c)) = c$ is non-null. The case where s and t are terms of the complete equality relationship graph with $[s] = [t]$ is by induction on the construction of the complete equality relationship graph, proving also the additional property that $Val(\alpha(t))$ is non-null for any term in the complete equality relationship graph.

Note that for all terms t of Q, we must have that $Val(\alpha(t))$ is non-null, since this is required for the atom in which t occurs to be satisfied under α. (There is one exception to this observation, the case in which t is the distinguished term and not equal to a variable. But then t must be an atomic value c, for which $Val(\alpha(t)) = c$ is non-null.) For equations $s = t$ in Q we must have $Val(\alpha(s)) = Val(\alpha(t))$ in order for α to be satisfying. It is trivial that for an equality edge $t = t$ we have $Val(\alpha(t)) = Val(\alpha(t))$. This establishes the base case of the induction. (The case of equations $t = t$, for terms introduced later in the construction, is similar, but uses the additional property.)

Consider next the case of edges $t_1 = t_2$ and $t_2 = t_3$ inducing an edge $t_1 = t_3$. Since we have $[t_1] = [t_2]$ and $[t_2] = [t_3]$ it follows from the inductive hypothesis that $Val(\alpha(t_i))$ is non-null for $i = 1 \ldots 3$ and $Val(\alpha(t_1)) = Val(\alpha(t_2))$ and $Val(\alpha(t_2)) = Val(\alpha(t_3))$. It is immediate that $Val(\alpha(t_1)) = Val(\alpha(t_3))$.

Finally, suppose that x and y are variables with $x = y$ an edge of the complete equality relationship graph, and that $x.A$ is a node of the complete equality relationship graph. By the induction hypothesis $Val(\alpha(x))$ and $Val(\alpha(y))$ are non-null and equal, and $Val(\alpha(x).A)$ is non-null. It follows that $Val(\alpha(y.A)) = Val(\alpha(y).A)$ is equal to $Val(\alpha(x).A)$ and hence non-null, and equal to $Val(\alpha(x.A))$.

Part (ii) follows directly from part (i) using the fact that the definition of satisfaction depends only on the values of the terms in the atoms $\mathbf{A}$ and $\mathbf{A}'$. $\square$

In addition to the atoms in a query, certain other atoms will always be satisfied with respect to any satisfying assignment for the query. For example, if a query contains atoms $x = y$ and $y \in C$ then every satisfying assignment also makes the atom $x \in C$ true. The following notion is intended to characterize such atoms. A query Q is said to *derive* an atom $\mathbf{A}$ if

1. $\mathbf{A}$ is of the form $t \in T$ where T is a basic type and $t \approx c$ for some atomic value c of type T, or
2. $\mathbf{A}$ is of the form $t_1 = t_2$, where t_1 and t_2 are terms satisfying $t_1 \approx t_2$, or
3. Q contains an atom $\mathbf{A}'$ such that $\mathbf{A}' \approx \mathbf{A}$.

We write $Q \vdash \mathbf{A}$ if Q derives the atom $\mathbf{A}$. The following result shows that every atom derived by a query in fact holds under any satisfying assignment.

Lemma 4.2 *Let α be any satisfying assignment for the query Q in the state* $\mathbf{s}$. *If* $\mathbf{A}$ *is an atom such that $Q \vdash \mathbf{A}$ then $\mathbf{s} \models \alpha(\mathbf{A})$.*

[Proof]: We consider each clause of the definition of derivation:

1. Suppose $\mathbf{A}$ is of the form $t \in T$ where T is an atomic type and $t \approx c$, where c is an atomic value of type T. By Lemma 4.1, we have $Val(\alpha(t)) = Val(\alpha(c)) = c$. It follows that $\mathbf{s} \models \alpha(\mathbf{A})$.
2. If $\mathbf{A}$ is of the form $s = t$ with $s \approx t$ then by Lemma 4.1(i), $Val(\alpha(s)) = Val(\alpha(t))$, with both non-null, so $\mathbf{s} \models \alpha(\mathbf{A})$ by definition.

3. Otherwise, Q contains an atom $\mathbf{A}'$ such that $\mathbf{A}' \approx \mathbf{A}$. Since α is a satisfying assignment, we have that $\mathbf{s} \models \alpha(\mathbf{A}')$. By Lemma 4.1(ii), it follows that $\mathbf{s} \models \alpha(\mathbf{A})$ also. $\Box$

We now set about showing that a converse to this result holds for the state $\mathbf{s}_Q$ constructed form a query Q. We begin by relating atoms over this state to atoms of the query. Define an *inverse* of the canonical mapping α for the query Q to be a function ω mapping each object of $\mathbf{s}_Q$ and each atomic value to either a variables of Q or an atomic value, such that

1. for all atomic values c such that there exists a variable y of Q with $\alpha(y) = c$, we have that $\omega(c)$ is a variable with this property, and
2. for all atomic values c such that there exists no variable y of Q with $\alpha(y) = c$, we have $\omega(c) = c$, and
3. for all objects $[t]$ of $\mathbf{s}_Q$ we have that $\omega([t])$ is a *variable* in $[t]$. (Note that each object of $\mathbf{s}_Q$ is an equivalence class of terms that must contain a variable by condition (ii) of the definition of well-formed queries.)

We may extend ω to a mapping from terms over the state $\mathbf{s}_Q$ to terms formed using the variables of Q by defining $\omega([t].A) = \omega([t]).A$ for all terms of the form $[t].A$. The following result explains why we call such a mapping an inverse of α. (To understand the condition on (ii), observe that ω is not defined on sets occurring in $\mathbf{s}_Q$.)

Lemma 4.3 *If ω is an inverse of the canonical mapping α from Q to $\mathbf{s}_Q$ then:*

(i) For all terms t of Q we have $\omega(\alpha(t)) \approx t$.

(ii) For all terms t over $\mathbf{s}_Q$ for which $Val(t)$ is non-null and not equal to a set, we have $\omega(Val(t)) \approx \omega(t)$.

(iii) For all terms t over $\mathbf{s}_Q$ for which $Val(t)$ is non-null, $\omega(t)$ is either an atomic value or a term in the complete equality relationship graph of Q.

[Proof]: For (i) we consider three cases, according as whether $\alpha(t)$ is an atomic value, object or attribute term.

Consider first the case where $\alpha(t)$ is the atomic value c. Note that t cannot be an attribute term since these must be mapped to attribute terms. If t is an atomic value, t must be equal to c, so $\omega(\alpha(t)) = c \approx c = t$. If t is a variable, there exists a variable y with $\alpha(y) = c$ and $\omega(c) = y$. Since the construction guarantees that distinct equivalence classes are mapped by α to distinct values, we must have $t \in [y]$. Thus $\omega(\alpha(t)) = y \approx t$.

Next, consider the case in which $\alpha(t)$ is the object $[x]$. As this case can only arise from case Val3 of the construction, we have $t \in [x]$, so $\omega(\alpha(t)) = \omega([x]) \approx t$.

Finally, if $\alpha(t)$ is of the form $o.A$, where o is an object of $\mathbf{s}_Q$, then t is a term of the form $x.A$, where x is a variable, and $o = [x]$. Thus $\omega(\alpha(t)) = \omega([x].A) = \omega([x]).A$. Let $\omega([x])$ be the variable y. Because y must be in $[x]$ and $x.A$ is a term of the complete equality graph, $y.A$ is also a term of the complete equality graph, with $[x.A] = [y.A]$. Hence we have that $\omega(\alpha(t)) = y.A \approx x.A = t$.

We prove (ii) and (iii) together. Note that if t is an atomic value or an object of $\mathbf{s}_Q$ then $\omega(t)$ is a variable of Q or an atomic value, so the claim of (iii) holds. In this case we also have $Val(t) = t$ so the claim of (ii) follows directly from the fact that $\omega(t)$ is an atomic value or a term of the complete equality relationship graph, so that $\omega(t) \approx \omega(t)$.

Suppose next that t is a term over $\mathbf{s}_Q$ of the form $[x].A$ (where, without loss of generality, x is a variable) for which $Val(t)$ is defined and not equal to a set. By definition, $\omega([x].A) = \omega([x]).A = y.A$ for some variable $y \in [x]$. Moreover, by construction of $\mathbf{s}_Q$, there exists a variable $z \in [x]$ such that $z.A$ is a term in the complete equality relationship graph. It follows that $\omega([x].A)$ are in the complete equality relationship graph, establishing the claim of (iii) in this case. (Note also that $x.A$ is in the complete equality relationship graph.)

Moreover, we have either $Val(t) = [x.A]$ or $Val(t) = c$ for some atomic value c. To complete the proof of (ii) we consider each of these cases individually.

Suppose first that $Val(t) = [x.A]$. Let $\omega([x])$ be the variable $y \in [x]$. Then $x \approx y$. Hence $\omega(Val(t)) = \omega([x.A]) \approx x.A \approx y.A = \omega(t)$. Next, if $Val(t)$ is the atomic value c then, by construction of $\mathbf{s}_Q$, one of the following cases applies:

1. In case (Val1) of the construction, we have $c \in [x.A]$. Thus $\omega(Val(t)) = \omega(c)$. If there

does not exist a variable v with $\omega(v) = c$, then $\omega(c) = c \approx x.A$. If there does exist such a variable, and $\omega(c) = v$, then we must have $v \in [x.A]$. Hence here $\omega(c) = v \approx x.A$ also. In either case, it follows that $\omega(\mathit{Val}(t)) \approx \omega(t)$.

2. In case (Val2) of the construction, $x.A$ is of atomic type, but $[x.A]$ contains no atomic value. In this case, there exists a variable y in $[x.A]$, for which we must have $\alpha(y) = c$. Without loss of generality, take y to be the variable such that $\omega(c) = y$. Then $\omega([x].A) = \omega([x]).A \approx x.A \approx y = \omega(\mathit{Val}(t))$. $\Box$

We may extend an inverse ω of α to a mapping from atoms $\mathbf{A}$ over $\mathbf{s}_Q$ to atoms over the variables of Q by defining $\omega(\mathbf{A})$ to be the atom obtained by substituting for each term t over $\mathbf{s}_Q$ in $\mathbf{A}$ the term $\omega(t)$.

We now show that the atoms derived by a query Q completely capture the set of atoms of a certain form holding in the state $\mathbf{s}_Q$. Define an atom over $\mathbf{s}_Q$ to be *terminal* if it is of one of the following forms:

1. $t \in T$, where t is a term over $\mathbf{s}_Q$ and T is an atomic type or terminal class,

2. $s = t$, where s and t are terms over $\mathbf{s}_Q$ such that neither $\mathit{Val}(s)$ nor $\mathit{Val}(s)$ is a set,

3. $t \in o.A$, where t is a term over $\mathbf{s}_Q$ and o is an object of $\mathbf{s}_Q$.

Intuitively, the terminal atoms are those that may occur as images under satisfying assigments of a well-formed query.

Lemma 4.4 *Let ω be any inverse of α. Then for all* terminal *atoms A over $\mathbf{s}_Q$ such that $\mathbf{s}_Q \models \mathbf{A}$ we have $Q \vdash \omega(\mathbf{A})$.*

[Proof]: We consider each of the possible cases for the atom A.

1. If $\mathbf{A}$ is of the form $c \in T$ where c is an atomic value of type T then $Q \vdash \omega(\mathbf{A})$ by the first clause of the definition of derivation.

2. Suppose $\mathbf{A}$ is of the form $[t] \in C$ where C is a terminal class and $[t]$ is an object of $\mathbf{s}_Q$, with $I_c([t]) = C$. By construction of $\mathbf{s}_Q$, there exists a variable $x \in [t]$ such that $x \in C$ is an

atom of Q. Since $\omega([t])$ is also an element of the equivalence class $[t]$ we have $x \approx \omega([t])$, so the atom $\omega(A)$ is $\approx$-related to the atom $x \in C$. Hence $Q \vdash \omega(\mathbf{A})$ by the third clause of the definition of derivation.

3. Suppose that $\mathbf{A}$ is of the form $s = t$, where s and t are terms over $\mathbf{s}_Q$. For this equation to hold in $\mathbf{s}_Q$, we must have $Val(s) = Val(t)$, with both non-null. Since the atom is terminal, neither value is a set. Hence, by Lemma 4.3(ii), we have $\omega(s) \approx \omega(Val(s)) = \omega(Val(s)) \approx \omega(t)$. It follows that $Q \vdash \omega(s) = \omega(t)$ by the second clause of the definition of derivation.

4. Suppose that $\mathbf{A}$ is an atom of the form $a \in b.A$, where a is a value or object of $\mathbf{s}_Q$ and b is an object of $\mathbf{s}_Q$. By construction of $\mathbf{s}_Q$, for this atom to hold in $\mathbf{s}_Q$ there must exist an object term t in Q and a set term $x.A$ such that $t \in x.A$ is an atom of Q and $\alpha(t) = a$ and $\alpha(x) = b$. Since $\omega(a) = \omega(\alpha(t)) \approx t$ and $\omega(b) = \omega(\alpha(x)) \approx x$ by Lemma 4.3 (i), it follows the third clause of the definition of derivation that $Q \vdash \omega(t) \in \omega(b).A$, i.e., $Q \vdash \omega(\mathbf{A})$. $\square$

We comment that the corresponding property could not be established for atoms over $\mathbf{s}_Q$ expressing equations between sets. For example, for the query $Q = \{x \mid 1 \in x.A \wedge 1 \in y.B\}$ we find that in $\mathbf{s}_Q$ we have $\mathbf{s}_Q \models [x].A = [y].B$, since $Val([x].A) = \{1\} = Val([y].B)$, but not $Q \vdash x.A = y.B$, since $[x.A] \neq [y.B]$.

We are now ready to state the characterization of containment of queries. First, define a *variable mapping* from a query Q_2 to a query Q_1 to be a function μ mapping each variable of Q_2 to either a variable of Q_1 or to an atomic value. Such a mapping can be extended to a mapping from terms in the complete equality graph of Q_2 to expressions formed from atomic values and variables of Q_1 by taking $\mu(c) = c$ for all atomic values c, and $\mu(x.B) = \mu(x).B$ for all variables x. (Note that the expression $\mu(x).B$ need not be a term of the complete equality graph of Q_1, or even a term. For example, if $\mu(x)$ is the atomic value c then this expression is $c.B$, which is not a term, and is uninterpretable in our language, since atomic values do not have attributes.)

We will deal with the composition of various such mappings. Recall that if $f : X \to Y$ and $g : Y \to Z$ are functions then the composition $g \circ f$ is the function from X to Z defined by

$g \circ f(x) = g(f(x))$.

Define a *containment mapping* μ from a query Q_2 to a query Q_1 to be a variable mapping from Q_2 to Q_1 such that

1. if t_1 is the distinguished term of Q_1 and t_2 is the distinguished term of Q_2 then $\mu(t_2) \approx_1 t_1$ (where $\approx_1$ is the relation derived from Q_1), and

2. for every atom A of Q_2, we have $Q_1 \vdash \mu(A)$.

The following result shows that containment mappings characterize containment between terminal conjunctive queries.

Theorem 4.5 *If Q_1 and Q_2 are terminal conjunctive queries then $Q_1 \subseteq Q_2$ if and only if there exists a containment mapping from Q_2 to Q_1.*

[Proof]: Suppose first that μ is a containment mapping from Q_2 to Q_1. We need to show that $Q_1 \subseteq Q_2$. For this, let $\mathbf{s}$ be any state and suppose that α is a satisfying assignment for Q_1 in the state $\mathbf{s}$, so that $\alpha(t_1) \in Q_1(\mathbf{s})$, where t_1 is the distinguished term of Q_1. We show that $\alpha(t_1)$ is also in $Q_2(\mathbf{s})$. Define the assignment β for Q_2 in $\mathbf{s}$ by $\beta = \alpha \circ \mu$. If $\mathbf{A}$ is an atom of Q_2 then since μ is a containment mapping we have by definition of containment that $Q_1 \vdash \mu(\mathbf{A})$. By Lemma 4.2 it follows that $\mathbf{s} \models \alpha(\mu(\mathbf{A}))$, i.e. that $\mathbf{s} \models \beta(\mathbf{A})$. Since this holds for every atom of Q_2 it follows that β is a satisfying assignment of Q_2 in $\mathbf{s}$. Thus $Q_2(\mathbf{s})$ contains $\beta(t_2)$, where t_2 is the distinguished term of Q_2. Because μ is a containment mapping, we have $\mu(t_2) \approx t_1$. Thus, by Lemma 4.1 we have that $\beta(t_2) = \alpha(\mu(t_2)) = \alpha(t_1)$ is in $Q_2(\mathbf{s})$, as promised. This completes the proof that $Q_1 \subseteq Q_2$, establishing the implication from right to left in the lemma.

To prove the converse, assume that there exists no containment mapping from Q_2 to Q_1. We show that Q_1 is not contained in Q_2 by establishing that $Q_1(\mathbf{s}_{Q_1})$ is not a subset of $Q_2(\mathbf{s}_{Q_1})$. In particular, we argue that if α is the canonical assignment of Q_1 in $\mathbf{s}_{Q_1}$ and and t_1 is the distinguished term of Q_1, then $\alpha(t_1)$ is not in $Q_2(\mathbf{s}_{Q_1})$. Note that, on the other hand, $\alpha(t_1)$ is an element of $Q_1(\mathbf{s}_{Q_1})$ by Lemma 3.2.

To show that $\alpha(t_1)$ is not in $Q_2(\mathbf{s}_{Q_1})$, assume to the contrary that β is a satisfying assignment for Q_2 in $\mathbf{s}_{Q_1}$ with $\beta(t_2) = \alpha(t_1)$, where t_2 is the distinguished term of Q_2. We derive a contradiction to the assumption that there exists no containment mapping from Q_2 to Q_1. In particular, let ω be any inverse of α and consider the mapping $\mu = \omega \circ \beta$. Note that this must be a variable mapping from Q_2 to Q_1. We claim that μ is a containment mapping from Q_2 to Q_1.

To see this, note first that $\mu(t_2) = \omega(\beta(t_2)) = \omega(\alpha(t_1)) \approx t_1$. Thus μ satisfies the first clause of the definition of containment mapping. Next, note that if $\mathbf{A}$ is an atom of Q_2 then since β is a satisfying assignment we have that $\mathbf{s}_{Q_1} \models \beta(\mathbf{A})$. Since $\beta(\mathbf{A})$ is a terminal atom over $\mathbf{s}_{Q_1}$ we have by Lemma 4.4 that $Q_1 \vdash \omega(\beta(\mathbf{A}))$, i.e., that $Q_1 \vdash \mu(\mathbf{A})$. Thus, μ satisfies the second condition of the definition of containment mapping, completing the proof that μ is a containment mapping from Q_2 to Q_1, and yielding the desired contradiction. $\Box$

5 Minimization of Terminal Conjunctive Queries

In this section, we define a notion of minimality of terminal conjunctive queries, and derive an algorithm that, given a terminal conjunctive query as input, finds a minimal equivalent query among all terminal conjunctive queries.

Let Q be a terminal conjunctive query. A *minimal* terminal conjunctive query of Q is a terminal conjunctive query equivalent to Q with the number of variables minimal among such terminal conjunctive queries. We now show how to find minimal queries.

We begin with a number of lemmas concerning containment mappings. In the rest of the discussion, we use subscripting to indicate the query with respect to which we compute the equivalence classes.

Lemma 5.1 *Let μ be a containment mapping from the satisfiable terminal conjunctive query Q_2 to the satisfiable terminal conjunctive query Q_1.*

(i) If t is a term of the complete equality relationship graph of Q_2, then $\mu(t)$ is either an atomic value or a term of the complete equality relationship graph of Q_1.

(ii) Let $\approx_1$ and $\approx_2$ be the relations on terms corresponding to the queries Q_1, Q_2, respectively. For all terms s and t, if $s \approx_2 t$ then $\mu(s) \approx_1 \mu(t)$.

[Proof]: We establish (i) and (ii) simultaneously. If $s \approx_2 t$ because both s and t are the atomic value c, then we have $\mu(s) = \mu(t) = c$, so $\mu(s) \approx_1 \mu(t)$. It therefore remains to show that if $[s]_2 = [t]_2$, where s and t are terms of the complete equality relationship graph of Q_2, then $\mu(s)$ and $\mu(t)$ are terms of the complete equality relationship graph of Q_1 and $[\mu(s)]_1 = [\mu(t)]_1$. We prove this by induction on the construction of the complete equality relationship graph of query Q_2. Note that it is immediate from the fact that μ is a containment mapping that for all terms s of Q such that $\mu(s)$ is not an atomic value, we have $\mu(s)$ equal to a term in the complete equality relationship graph of Q_1. This is because s occurs in an atom $\mathbf{A}$ of Q_2 and Q_1 derives the atom $\mu(\mathbf{A})$.

In the case of edges $s = s$, we clearly have $\mu(s) \approx_1 \mu(s)$. For edges $s = t$ corresponding to atoms of Q_2, we have $\mu(s) \approx_1 \mu(t)$ because μ is a containment mapping. Suppose that an edge $s = u$ is derived from edges $s = t$ and $t = u$ of the complete equality relationship graph of Q_2 for which we have $\mu(s) \approx_1 \mu(t)$ and $\mu(t) \approx_1 \mu(u)$. Since all the latter terms are in Q_1, it follows from the fact that $\approx_1$ is an equivalence relation on the terms of Q_1 that $\mu(s) \approx_1 \mu(u)$.

Finally, suppose that an edge $x.A = y.A$ is derived from an edge $x = y$ and a term $x.A$ of the complete equality relationship graph of Q_2. We assume by way of induction that $\mu(x) \approx_1 \mu(y)$ and that the term $\mu(x.A)$ occurs in the complete equality relationship graph of Q_1. Now $\mu(x.A) = \mu(x).A$, so $\mu(x)$ must be a variable, else Q_1 would not be satisfiable. Since $\mu(x) \approx_1 \mu(y)$, it follows similarly that $\mu(y)$ must be a variable. Thus, $\mu(y.A) = \mu(y).A$ is a term of the complete equality relationship graph of Q_1 and $\mu(x).A \approx_1 \mu(y).A$, by the inductive hypothesis and the construction of the complete equality relationship graph of Q_1. $\Box$

Lemma 5.2 *Let μ be a containment mapping from the satisfiable terminal conjunctive query Q_2 to the satisfiable terminal conjunctive query Q_1. If $\mathbf{A}$ is an atom such that $Q_1 \vdash \mathbf{A}$ then $Q_2 \vdash \mu(\mathbf{A})$.*

[Proof]: We consider the three cases of the definition of derivation. First, suppose $\mathbf{A}$ is of the form $t \in T$ where T is an atomic type and t is $\approx$-related to the atomic value c of type T. Then by Lemma 5.1 (ii) we have $\mu(t) \approx_1 c$, so $Q_1 \vdash \mu(t) \in T$. Second, if $\mathbf{A}$ is the atom $s = t$ and $s \approx_2 t$, then by Lemma 5.1 (ii) we have $\mu(s) \approx_1 \mu(t)$, so $Q_1 \vdash \mu(s) = \mu(t)$. Finally, if Q_2 contains the atom $\mathbf{A}' \approx_2 \mathbf{A}$ then by Lemma 5.1 (ii) we have $\mu(\mathbf{A}') \approx_1 \mu(\mathbf{A})$. Since μ is a containment mapping it is also the case that $Q_1 \vdash \mu(\mathbf{A}')$. It follows using the definition of derivation and the fact that $\approx_1$ is an equivalence relation on terms of the complete equality relationship graph that $Q_1 \vdash \mu(\mathbf{A})$. $\Box$

Lemma 5.3 *Let Q_1, Q_2 and Q_3 be satisfiable terminal conjunctive queries. If μ_1 is a containment mapping from Q_1 to Q_2 and μ_1 is a containment mapping from Q_2 to Q_3 then $\mu_2 \circ \mu_1$ is a containment mapping from Q_1 to Q_3.*

[Proof]: Let t_1, t_2 and t_3 be the distinguished terms of Q_1, Q_2 and Q_3, respectively, and let $\approx_1, \approx_2$ and $\approx_3$ be the relations on terms derived form these queries. Since μ_1 and μ_2 are containment mappings, we have $\mu_1(t_1) \approx_2 t_2$ and $\mu_2(t_2) \approx_3 t_3$. It follows using Lemma 5.1(ii) that $\mu_2 \circ \mu_1(t_1) \approx_3 \mu_2(t_2) \approx_3 t_3$. This establishes that $\mu_2 \circ \mu_1$ satisfies the first condition of the definition of containment mapping.

It remains to show that if $\mathbf{A}$ is an atom of Q_1 then $Q_3 \vdash \mu_2 \circ \mu_1(\mathbf{A})$. Since μ_1 is a containment mapping we have that $Q_2 \vdash \mu_1(\mathbf{A})$. It follows using Lemma 5.2 and the fact that μ_2 is a containment mapping that $Q_3 \vdash \mu_2 \circ \mu_1(\mathbf{A})$. $\Box$

Let $Q = \{t \mid M\}$ be a conjunctive query and suppose μ is a variable mapping on Q. Define $\mu(Q)$ to be the conjunctive query with the distinguished term $\mu(t)$ obtained by replacing each atom $\mathbf{A}$ of Q by the atom $\mu(\mathbf{A})$.

Proposition 5.4 *Let Q be a terminal conjunctive query. Suppose there is a containment mapping μ from Q to itself. Then $\mu(Q)$ is equivalent to Q.*

[Proof]: It is easy to check that μ is a containment mapping from Q to $\mu(Q)$, so we have by Theorem 4.5 that $\mu(Q) \subseteq Q$. To show $Q \subseteq \mu(Q)$, we show that there is a containment mapping from $\mu(Q)$ to Q. We claim that the identity mapping i is such a mapping. Note first that the distinguished term of $\mu(Q)$ is $\mu(t)$, where t is the distinguished term of Q, and we have $i(\mu(t)) = \mu(t) \approx t$ because μ is a containment mapping. It remains to show that for every atom $\mathbf{A}$ of Q, we have for the corresponding atom $\mu(\mathbf{A})$ of $\mu(Q)$ that $Q \vdash i(\mu(\mathbf{A}))$. That is, we need $Q \vdash \mu(\mathbf{A})$. This is immediate from the fact that μ is a containment mapping. $\Box$

The following describes how to obtain a minimal terminal conjunctive query. Say that a variable mapping μ from a query Q_1 to a query Q_2 is *bijective* if $\mu(x)$ is a variable of Q_2 for every variable x of Q_1, and the restriction of μ to the set of variables of Q_1 is a bijective mapping to the set of variables of Q_2.

Theorem 5.5 *A terminal conjunctive query Q is minimal if all containment mappings from Q to itself are bijective.*

[Proof]: Suppose that all containment mappings from Q to itself are bijective, but that Q is not minimal. We establish a contradiction. Since Q is not minimal, there is a minimal terminal conjunctive query Q' equivalent to Q which has fewer variables. Now by Theorem 4.5, the fact that $Q' \equiv Q$ implies that there is a containment mapping μ from Q to Q' and also that there is a containment mapping ω from Q' to Q. It follows from Lemma 5.3 that the composite mapping $\omega \circ \mu$ is a variable mapping from Q to itself. By the assumption, $\omega \circ \mu$ is bijective. But this means that the number of variables of Q' is at least as large as the number of variables of Q, contradicting the choice of Q'. $\Box$

The converse to Theorem 5.5 follows from the following.

Theorem 5.6 *Let Q_1 and Q_2 be minimal terminal conjunctive queries. Suppose $Q_1 \equiv Q_2$. Then every containment mapping from one query to the other is bijective.*

[Proof]: Let μ be a containment mapping from Q_1 to Q_2. Since these queries are equivalent, there also exists a containment mapping ω from Q_2 to Q_1. By Lemma 5.3, the composite mapping $\mu \circ \omega$ is a containment mapping from Q_2 to itself. Suppose that the image of the set of variables of Q_2 under $\mu \circ \omega$ does not contain all the variables of Q_2. Then the query $(\mu \circ \omega)(Q_2)$, which is equivalent to Q_2 by Proposition 5.4, has fewer variables than Q_2. This contradicts the minimality of Q_2. This shows that the image of the the set of variables of Q_2 under $\mu \circ \omega$ contains all the variables of Q_2. It follows that Q_1 has at least as many variables as Q_2. A similar argument using the containment mapping $\omega \circ \mu$ from Q_1 to Q_1 shows that Q_2 has at least as many variables as Q_1. Since $\mu \circ \omega$ covers the variables of Q_2, it now follows that μ must in fact be a bijection between the variables of Q_1 and Q_2. $\Box$

Given a satisfiable terminal conjunctive query Q the algorithm to find a minimal terminal conjunctive query Q' equivalent to Q is as follows. Consider all containment mappings μ from Q to itself. Choose Q' to be one of the queries $\mu(Q)$ that has the fewest variables among such queries. The minimality of Q' follows from Theorem 5.6.

Note that there may be some further optimizations possible for the query Q' so obtained, since this may contain atoms of the form $c \in T$ or $c = c$, where c is an atomic value of type T. Such atoms, since they are derivable even from an empty query, can be deleted, yielding an equivalent query.

6 Containment and Optimization of Conjunctive Queries

In this section, we study the containment of conjunctive queries and show how to obtain optimal conjunctive queries. The optimal conjunctive queries obtained are expressed as unions of terminal conjunctive queries and are optimal among all unions of terminal conjunctive queries. In Section 6.1, we characterize containment of conjunctive queries by solving the containment problem for unions of terminal conjunctive queries. In Section 6.2, we give our notion of optimality. In Section 6.3, we derive an algorithm, given a conjunctive query, for finding an optimal union of terminal conjunctive queries. We first use an example to illustrate our notion of optimality and the approach taken in obtaining an optimal query.

Example 6.1 *Let us consider the following query defined on the schema in Example 1.1.*

Q_1: $\{\ x \mid \exists y\, \exists z\ (x{\in} \mathit{Vehicle}\ \&\ y{\in}\mathit{Discount}\ \&\ z{\in}\mathit{Client}\ \&\ x{\in}y.\mathit{VehRented}\ \&\ x{\in}z.\mathit{VehRented})\}$.

This query retrieves all those vehicles that have been rented to a discount client. By Proposition 2.1, Q_1 is equivalent to the union of the following terminal conjunctive queries:

S_1: $\{\ x \mid \exists y\, \exists z\ (x{\in}\mathit{Auto}\ \&\ y{\in}\mathit{Discount}\ \&\ z{\in}\mathit{Normal}\ \&\ x{\in}y.\mathit{VehRented}\ \&\ x{\in}z.\mathit{VehRented})\}$.

S_2: $\{\ x \mid \exists y\, \exists z\ (x{\in}\mathit{Auto}\ \&\ y{\in}\mathit{Discount}\ \&\ z{\in}\mathit{Discount}\ \&\ x{\in}y.\mathit{VehRented}\ \&\ x{\in}z.\mathit{VehRented})\}$.

S_3: $\{\ x \mid \exists y\, \exists z\ (x{\in}\mathit{Trailer}\ \&\ y{\in}\mathit{Discount}\ \&\ z{\in}\mathit{Normal}\ \&\ x{\in}y.\mathit{VehRented}\ \&\ x{\in}z.\mathit{VehRented})\}$.

S_4: $\{\ x \mid \exists y\, \exists z\ (x{\in}\mathit{Trailer}\ \&\ y{\in}\mathit{Discount}\ \&\ z{\in}\mathit{Discount}\ \&\ x{\in}y.\mathit{VehRented}\ \&\ x{\in}z.\mathit{VehRented})\}$.

S_5: $\{\ x \mid \exists y\, \exists z\ (x{\in}\mathit{Truck}\ \&\ y{\in}\mathit{Discount}\ \&\ z{\in}\mathit{Normal}\ \&\ x{\in}y.\mathit{VehRented}\ \&\ x{\in}z.\mathit{VehRented})\}$.

S_6: $\{\ x \mid \exists y\, \exists z\ (x{\in}\mathit{Truck}\ \&\ y{\in}\mathit{Discount}\ \&\ z{\in}\mathit{Discount}\ \&\ x{\in}y.\mathit{VehRented}\ \&\ x{\in}z.\mathit{VehRented})\}$.

With algorithm SatTestUT, it can be shown that S_3, S_4, S_5 and S_6 are unsatisfiable. The reason for this is that discount clients are only allowed to rent automobiles and not other types of vehicles. Hence Q_1 is equivalent to $S_1 \cup S_2$. There is a containment mapping from S_2 to S_1; the mapping is to map x to x and y and z to y. By Theorem 4.5, S_1 is redundant and is removed from the union. S_2 can further be minimized by mapping x to x and y and z to y. The resulting optimal query obtained is:

S_2': $\{\ x \mid \exists y$ ($x \in$Auto & $y \in$Discount & $x \in y$.VehRented)$\}$. $\square$

6.1 A Characterization of Containment of Unions of Terminal Conjunctive Queries

By Proposition 2.1, understanding containment of unions of terminal conjunctive queries suffices to solve the containment problem of conjunctive queries. We have found a characterization of containment for terminal conjunctive queries. We are now ready to state the containment condition for two unions of terminal conjunctive queries.

Theorem 6.1 *Let $M = Q_1 \cup \cdots \cup Q_s$ and $N = P_1 \cup \cdots \cup P_t$ be two unions of terminal 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]: "If" Trivial.

"Only if" Let Q_i be a subquery in M. Let $\mathbf{s}_{Q_i}$ be the state constructed for Q_i. Suppose α is a satisfying assignment from Q_i to $\mathbf{s}_{Q_i}$ and let ω be an inverse of α. Since $M \subseteq N$, there is some P_j such that there is a satisfying assignment γ of object identifiers and atomic values in the state $\mathbf{s}_{Q_i}$ to variables in P_j that gives rise to the answer $\alpha(t_i)$, where t_i is the distinguished term of Q_i. Then $\omega o \gamma$ is a variable mapping from P_j to Q_i. We show that the mapping $\omega o \gamma$ is a containment mapping.

Let $\omega o \gamma(t_j) = v$, where t_j is the distinguished term of P_j. Then $\gamma(t_j) = \alpha(t_i)$. Hence, $\omega(\gamma(t_j)) = \omega(\alpha(t_i))$. By Lemma 4.3(i), $\omega(\alpha(t_i)) \approx t_j$. Let $\mathbf{A}$ be an atom of P_j. Then $\gamma(\mathbf{A})$ is a terminal atom over $\mathbf{s}_{Q_i}$ and $\mathbf{s}_{Q_i} \models \gamma$ ($\mathbf{A}$). By Lemma 4.4, $Q_i \vdash \omega$ (γ ($\mathbf{A}$)). Hence $\omega o \gamma$ is a containment mapping. By Theorem 4.5, $Q_i \subseteq P_j$. $\square$

As a corollary, we solve the problem of determining when one conjunctive query contains the other one.

6.2 Search-Space-Optimal Queries

We now introduce our notion of optimality which intends to capture the intuition that the number of variables as well as their search spaces are minimal among all equivalent queries. In a conjunctive query, each variable is associated with a set of terminal classes or atomic types which denotes the search space of the variable. Without knowing the physical data organization for various classes, a good criterion of evaluating various equivalent queries is by comparing the set of variables in a query and their associated search spaces.

Example 6.2 *Let us consider again the Vehicle Rental Schema. The following three queries can be shown to be equivalent.*

Q_1: $\{ x \mid \exists y\, \exists z\ \ (x \in Auto\ \&\ y \in Discount\ \&\ z \in Vehicle\ \&\ x \in y.VehRented\ \&\ z \in y.VehRented)\}$.

Q_2: $\{ x \mid \exists y\ \ (x \in Vehicle\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$.

Q_3: $\{ x \mid \exists y\ (x \in Auto\ \&\ y \in Discount\ \&\ x \in y.VehRented)\}$.

If we consider the domain of the type of a variable as its search space, then Q_1 has more variables and has a larger search space than Q_2. Although Q_2 and Q_3 have the same number of variables, the search space associated with variables in Q_2 is greater than that in Q_3. Q_3 is considered to be more optimal since the number of variables as well as the search space are minimal. □

Existing work on exact minimization tries to minimize the number of joins in an expression [3]. The notion of optimality we shall propose attempts to generalize that idea.

A *multiset* is a set or bag of elements in which *duplicate* elements are allowed. Let S and T be two multisets. The *bag union* of S and T, denotes $S \uplus T$, is a multiset obtained from merging elements in the operands such that for every element x in S or T, the number of occurrences of x in the bag union is the sum of the numbers of occurrences of x in S and in T. Clearly the bag union operator is commutative and associative. We say S is a *bag subset* of T, denotes S

$\sqsubseteq T$, if for every element x in S, if there are n occurrences of x in S, then there are at least n occurrences of x in T.

Let Q be a conjunctive query and x a variable in Q. Define *term-class*$(Q,\ x)$ = $\{E|\ x \in C_1 \sqcup \cdots \sqcup C_n$ is the range atom in Q associated with the variable x, and E is a terminal subtype of C_i, for some $1 \leq i \leq n\}$. Informally, *term-class*$(Q,\ x)$ gives the terminal descendent classes or atomic types over which the variable x is ranging in the query. Let x_1 , ... , x_n be the set of variables in Q. Then *term-class*(Q) is a multiset defined as *term-class*$(Q,\ x_1) \uplus \ldots \uplus$ *term-class*$(Q,\ x_n)$.

We are now ready to define our notion of optimality. Let $Q{=}Q_1 \cup \cdots \cup Q_n$ and $P{=}P_1 \cup \cdots \cup P_m$ be two unions of conjunctive queries. Q is said to be *at least as optimal* as P, denotes $Q \leq P$, if *term-class*$(Q_1) \uplus \ldots \uplus$ *term-class*$(Q_n) \sqsubseteq$ *term-class*$(P_1) \uplus \ldots \uplus$ *term-class*(P_m).

A query Q is *search-space-optimal* among a set of queries $\mathbf{S}$ if for all P in $\mathbf{S}$ such that P is equivalent to Q, $P \leq Q$ implies $Q \leq P$. For search-space-optimal queries, the object search spaces are minimal among all equivalent queries in the set $\mathbf{S}$. The query Q_3 in Example 6.2 is a search-space-optimal query.

Corollary 6.2 *If Q is a minimal terminal conjunctive query, then Q is a search-space-optimal query among all terminal conjunctive queries.*

[Proof]: Follows from Theorem 5.6 and the derivability of range atoms. □

6.3 Optimization of Unions of Terminal Conjunctive Queries

In this subsection, we study the optimization of unions of terminal conjunctive queries. We show how to obtain a search-space-optimal query among all unions of terminal conjunctive queries.

A union of terminal conjunctive 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 $Q_i \subseteq Q_j$. We can transform a union of terminal conjunctive queries to an equivalent nonredundant union by finding Q_i and Q_j, $i{\neq}j$, such that $Q_i \subseteq Q_j$ and deleting Q_i from the union until no more subquery can be removed.

The following is an important property about nonredundant unions of terminal conjunctive queries.

Theorem 6.3 *Let $M = Q_1 \cup \cdots \cup Q_s$ and $N = P_1 \cup \cdots \cup P_t$ be two nonredundant unions of terminal conjunctive queries. Then $M{\equiv}N$ 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]: "If" Follows from Theorem 6.1.

"Only if" Suppose $M{\equiv}N$. Let Q_i be a subquery in M. By Theorem 6.1, there is a P_j in N such that $Q_i{\subseteq}P_j$. By the assumption on equivalence and by Theorem 6.1, there is Q_p in M such that $P_j{\subseteq}Q_p$. If $i{\neq}p$, then $Q_i{\subseteq}P_j$ and $P_j{\subseteq}Q_p$. This implies $Q_i{\subseteq}Q_p$ and contradicts the nonredundancy of M. Hence $i{=}p$ and $Q_i{\equiv}P_j$. If $s{\neq}t$, say, $s{<}t$, then there is a P_j which is redundant. This is a contradiction. It follows that $s{=}t$. □

The following is an algorithm for finding an optimal union of terminal conjunctive queries for a conjunctive query.

Algorithm Optimization: Given a conjunctive query Q, find an equivalent union of terminal conjunctive queries which is search-space-optimal among all unions of terminal conjunctive queries.

Input: A conjunctive query Q.

Output: An equivalent union of terminal conjunctive queries.

Method:

(1) Convert Q into an equivalent union of terminal conjunctive queries.

(2) Remove unsatisfiable subqueries from the union using the algorithm *SatTestUT*.

(3) Remove any redundant subqueries from the union.

(4) Minimize each of the remaining subqueries.

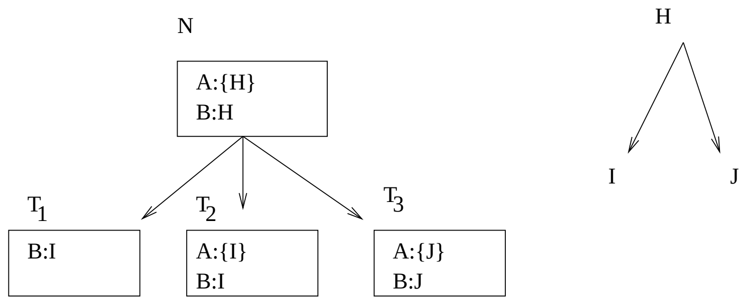
(5) Output the union of resulting terminal conjunctive queries.

Theorem 6.4 *The union of terminal conjunctive queries output by Algorithm Optimization is equivalent to Q and is a search-space-optimal query among all unions of terminal conjunctive queries.*

[Proof]: By Proposition 2.1, Theorem 3.3 and by definitions of redundancy and minimization, the union, say Q', output is equivalent to the input Q. Let $P= P_1 \cup \cdots \cup P_n$ be a union of terminal conjunctive queries that is equivalent to Q. Without loss of generality, let us assume that P is nonredundant and each subquery is minimal. By Theorem 6.3, there is a one-one correspondence between the two unions. By Theorem 5.6 and the fact that both unions are unions of terminal conjunctive queries, $Q' \leq P$. □

Let us look at the following example.

Example 6.3 *Let us consider a query defined on the following schema.*



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

By Proposition 2.1, Q_1 *is equivalent to the union of the following terminal conjunctive queries:*

S_1: $\{ x \mid \exists y\ \exists s\ (x \in T_1\ \&\ y \in I\ \&\ s \in J\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A) \}$.

S_2: $\{ x \mid \exists y\ \exists s\ (x \in T_2\ \&\ y \in I\ \&\ s \in J\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A) \}$.

S_3: $\{ x \mid \exists y\ \exists s\ (x \in T_3\ \&\ y \in I\ \&\ s \in J\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A) \}$.

S_4: $\{ x \mid \exists y\ \exists s\ (x \in T_1\ \&\ y \in J\ \&\ s \in J\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A) \}$.

S_5: $\{ x \mid \exists y\ \exists s\ (x \in T_2\ \&\ y \in J\ \&\ s \in J\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A) \}$.

S_6: $\{ x \mid \exists y\ \exists s\ (x \in T_3\ \&\ y \in J\ \&\ s \in J\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A) \}$.

With algorithm SatTestUT, S_2, S_3, S_4 *and* S_5 *are unsatisfiable. Hence* Q_1 *is equivalent to* $S_1 \cup S_6$. *Neither subquery in the union contains the other and hence the union is nonredundant. Since variables in* S_1 *range over different terminal classes,* S_1 *is minimal. It can easily be shown that* S_6 *can be minimized further. A minimal form is as follows:*

S_6': $\{ x \mid \exists y\ (x \in T_3\ \&\ y \in J\ \&\ y = x.B\ \&\ y \in x.A) \}$.

The union of terminal conjunctive queries output by the algorithm is $S_1 \cup S_6'$ which is a search-space-optimal query for Q_1 among all unions of terminal conjunctive queries. □

Algorithm *Optimization* only produces an optimal query expressed as a union of terminal conjunctive queries. This form needs not be the most desirable form to be executed. For instance, Q_1 in Example 6.3 is equivalent to the following query.

Q_2: $\{\ x \mid \exists y \exists s (x \in T_1 \sqcup T_3\ \&\ y \in H\ \&\ s \in J\ \&\ y = x.B\ \&\ y \in x.A\ \&\ s \in x.A)\}$.

Throughout the discussion, we made *no* assumption on how data are being physically organized. It could be the case that, given certain information on data organization, Q_2 is a better form to be evaluated than the union produced by the algorithm. However, the union of terminal conjunctive queries produced could be used as a basis to generate equivalent query in a more desirable form. It would be interesting to see how other information could be used to synthesize a more optimal query for the union.

7 Complexity of the Containment Problem

In this section, we investigate the time complexity for determining containment of conjunctive queries. We begin with the simple case of terminal conjunctive queries. The containment problem for terminal conjunctive queries is clearly in NP. A relational query is called a SPJ-query if only selection with constant, projection and natural join are used in the query. It is well-known that the class of relational SPJ-expressions can be expressed as a tagged tableau [3]. Every such tagged tableau can be translated into a conjunctive query without the set membership construct. In [3], it was shown that the problem of determining containment of SPJ-expressions is an NP-complete problem. Consequently, the containment problem of terminal conjunctive queries is also NP-complete. In fact, it can be shown that containment problem of terminal conjunctive queries involving only range and set membership atoms is NP-complete.

Corollary 7.1 *The problem of determining containment for terminal conjunctive queries is an NP-complete problem.*

[Proof]: Follows from the argument above. □

Corollary 7.1 implies that testing containment of conjunctive queries is NP-hard. We shall show that the problem is in Π_2^P of the polynomial hierarchy [32]. The proof of this result is similar to a proof in [28]. A language L is in Π_2^P if its complement is in Σ_2^P, the class of languages which can be recognized by nondeterministic polynomial-time algorithm with an oracle from NP. An *oracle* for a class of decision problems **C** enables one to decide any problem in **C** in unit time. The classes Σ_2^P and Π_2^P contain NP and are contained in PSPACE.

Theorem 7.2 *The problem of determining containment for conjunctive queries is in* Π_2^P.

[Proof]: Let E_1 and E_2 be two conjunctive queries. We describe a nondeterministic polynomial-time algorithm with an oracle from NP that answers "yes" if and only if $E_1 \not\subseteq E_2$. By Theorem 6.1, $E_1 \not\subseteq E_2$ if and only if there is a terminal conjunctive query in the union for E_1, say E_3, such that $E_3 \not\subseteq E_2$. E_3 can be guessed nondeterministically from E_1 by assigning terminal classes or atomic types to variables involved. After guessing the query, we test if E_3 is satisfiable. Testing satisfiability of terminal conjunctive queries can be performed in polynomial time [10]. If E_3 is satisfiable, then we ask the oracle if $E_3 \subseteq E_2$. If the oracle answers "no", then the algorithm answers "yes". It remains to show that determining $E_3 \subseteq E_2$ is in NP. By Theorem 6.1, $E_3 \subseteq E_2$ if and only if there is a terminal conjunctive query in the union for E_2, say E_4, such that $E_3 \subseteq E_4$. Thus we can guess E_4 as before and then check in nondeteministic polynomial time that $E_3 \subseteq E_4$. Hence our claim is proved. □

We are now ready to show that the containment problem is Π_2^P- hard.

A Π_2 formula of quantified propositional logic is an expression

$$\varphi = \forall p_1 \dots p_n \exists p_{n+1} \dots p_{n+m}[\alpha]$$

where α is a formula of propositional logic containing only the propositional variables $p_1, \dots, p_{n+m}$. Such an expression is true if for every assignment of boolean truth value to the variables $p_1, \dots, p_n$, there exists an assignment of truth values to the variables $p_{n+1}, \dots, p_{n+m}$ under which the formula α is true. The set Π_2-SAT is the set of all true Π_2 formulae. This is a generalization of the problem of satisfiability to the polynomial hierarchy. It is known that the set Π_2-SAT is complete for the level Π_2^p of this hierarchy [33, 36].

Theorem 7.3 *There exists a fixed schema* $\mathbf{S}$ *such that problem of deciding, given queries* Q_1 *and* Q_2 *on* $\mathbf{S}$*, whether* $Q_1 \subseteq Q_2$ *is* Π_2^p*-hard.*

[Proof]: By reduction from Π_2-SAT. We show that for every Π_2 formula φ, there is a pair of conjunctive queries Q_1, Q_2 such that φ is true if and only if Q_1 is contained in Q_2.

Define the schema $\mathbf{S}$ to contain classes C, R, G, V, AND and NOT. The subclass relationships between these classes are given by $R \prec C$ and $G \prec C$. Thus, the terminal classes are R, G, V, AND and NOT. The tuple type for both R and G is $[a : C,\ b : INT,\ c : V]$, for V is the empty tuple type $[]$, for AND is $[in_1 : V,\ in_2 : V,\ out : V]$, and for NOT is $[in : V,\ out : V]$.

We first describe the query Q_1. Part of this query will encode the truth tables for conjunction and negation. Intuitively, there are (four) variables ranging over AND which represent the lines of the truth table for '$\wedge$', and there are (two) variables ranging over NOT which represent the lines of the truth table of '$\neg$', and there are variables t and f in V which denote the truth values *true* and *false*, respectively. We write $TT(t, f)$ for the following formula:

$$\begin{array}{l}
\exists u_1 \in AND\ [u_1.in_1 = t\ \&\ u_1.in_2 = t\ \&\ u_1.out = t]\ \& \\
\exists u_2 \in AND\ [u_2.in_1 = t\ \&\ u_2.in_2 = f\ \&\ u_2.out = f]\ \& \\
\exists u_3 \in AND\ [u_3.in_1 = f\ \&\ u_3.in_2 = t\ \&\ u_3.out = f]\ \& \\
\exists u_4 \in AND\ [u_4.in_1 = f\ \&\ u_4.in_2 = f\ \&\ u_4.out = f]\ \& \\
\exists v_1 \in NOT\ [v_1.in = t\ \&\ v_1.out = f]\ \& \\
\exists v_2 \in NOT\ [v_2.in = f\ \&\ v_2.out = t].
\end{array}$$

Next, suppose φ has n universally quantified variables. Part of the query Q_1 will have the function of assigning a truth value to each of these variables. We construct for each $i = 1, \ldots, n$ the query $ASGN_i(t, f)$ given by

$$\begin{array}{llll}
\exists w_{i1} \in R\ \exists w_{i2} \in C\ \exists w_{i3} \in G\ [w_{i1}.a = w_{i2}\ \& & w_{i2}.a = w_{i3} & \& & \\
& w_{i2}.b = i & \&\ w_{i3}.b = i\ \& & \\
& w_{i2}.c = t & \&\ w_{i3}.c = f]. &
\end{array}$$

We now define the query Q_1 to be

$$\{t \mid t \in V\ \&\ \exists f \in V\ [TT(t, f)\ \&\ ASGN_1(t, f)\ \& \ldots \&\ ASGN_n(t, f)]\}.$$

After moving the quantifiers to the front, it is clear that Q_1 is a conjunctive query.

Let α be a formula of propositional logic in the propositional constants $p_1,\ldots,p_{n+m}$. We define inductively the formula Φ_α with free variables amongst $\mathbf{x} = x_{p_n},\ldots,x_{p_{n+m}}$ and x_α. If α is the propositional constant p_i, where $i = 1,\ldots,n$, then

$$\Phi_\alpha = \exists z_{i1} \in R \exists z_{i2} \in G[z_{i1}.a = z_{i2} \ \& \ z_{i2}.b = i \ \& \ z_{i2}.c = x_{p_i}].$$

If α is the propositional constant p_i, where $i = n+1,\ldots,n+m$, then Φ_α is the null formula **true**. (Note that the variable x_α is x_{p_i} in these cases.) If $\alpha = \beta \& \gamma$ then $\Phi_\alpha(\mathbf{x}, x_\alpha)$ is

$$\exists y_\beta \in AND \ \exists x_\beta x_\gamma \in V \ \left[\begin{array}{l} y_\beta.in_1 = x_\beta \ \& \ y_\beta.in_2 = x_\gamma \ \& \ y_\beta.out = x_\alpha \ \& \\ \Phi_\beta(\mathbf{x}, x_\beta) \ \& \ \Phi_\gamma(\mathbf{x}, x_\gamma) \end{array} \right]$$

If $\alpha = \neg\beta$ then $\Phi_\alpha(\mathbf{x}, x_\alpha)$ is

$$\exists y_\beta \in NOT \ \exists x_\beta \in V \ [x.in = x_\beta \& y_\beta.out = x_\alpha \& \Phi_\beta(\mathbf{x}, x_\beta)].$$

We define Q_2 to be the query

$$\{x_\alpha \mid x_\alpha \in V \ \& \ \exists x_{p_1} \ldots x_{p_n} \in V \ [\Phi_\alpha(\mathbf{x}, x_\alpha)]\}$$

Observe that the class C does not occur in Q_2. Thus, after moving the quantifiers to the front, this query is a terminal conjunctive query.

Note that expansions of Q_1 are obtained by replacing each of the n range atoms $w_{i2} \in C$ by either $w_{i2} \in R$ or $w_{i2} \in G$. There are therefore 2^n such expansions. We first show that each of these expansions uniquely determines an assignment of truth values to the propositional constants $p_1,\ldots,p_n$.

Suppose $1 \leq i \leq n$. Because the constant i has only two occurrences in Q_1, if μ is a mapping from the variables z_{i1}, z_{i2}, x_{p_i} to the variables of an expansion E of Q_1 that preserves the atom $z_{i2}.b = i$ of the formula Φ_{p_i}, then we must have $\mu(z_{i2}) = w_{i2}$ or $\mu(z_{i2}) = w_{i3}$. In case the atom $w_{i2} \in C$ of $ASGN_i$ is expanded as $w_{i2} \in G$, the mapping $z_{i1} \mapsto w_{i1}, z_{i2} \mapsto w_{i2}, x_{p_i} \mapsto t$, is the only mapping that preserves all the equality and range atoms of Φ_{p_i}. This determines the assignment of *true* to p_i. Similarly, in case $w_{i2} \in C$ is expanded as $w_{i2} \in R$ the mapping $z_{i1} \mapsto w_{i2}, z_{i2} \mapsto w_{i3}, x_{p_i} \mapsto f$ is the only such mapping. This determines the assignment of *false* to p_i.

Next, suppose that μ is a mapping from the variables of the formulae Φ_{p_i} for $i = 1,\ldots,n$ and the variables $x_{p_{n+1}},\ldots,x_{p_{n+m}}$ to the variables of an expansion E of Q_1, such that the atoms of Q_2 containing these variables are preserved. As noted above, this implies that the variables $x_{p_1},\ldots,x_{p_n}$ are mapped to either t or f. The same holds for the variables $x_{p_{n+1}},\ldots,x_{p_{n+m}}$, because of the range atoms $x_{p_i} \in V$ in Q_2. Let θ be the truth value assignment that assigns the constant p_i to be *true* if and only if $\mu(x_{p_i}) = t$. Under these conditions, a straightforward induction on the complexity of α shows that there exists a unique extension of the mapping μ to a mapping from the variables of Q_2 to the variables of Q_1 that preserves all the atoms of Q_2. Furthermore, we have $\mu(x_\alpha) = t$ if and only if the formula α is true with respect to the assignment θ. Note also that this mapping μ is a containment mapping from Q_2 to the expansion E if and only if $\mu(x_\alpha) = t$.

It now follows from the observations above that there exists a containment mapping from Q_2 to E for each expansion E of Q_1 if and only if the quantified formula φ is true. □

Theorem 7.4 *The problem of determining containment for conjunctive queries is complete in* Π_2^P.

[Proof]: By Theorems 7.2 and 7.3. □

8 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, equivalence and optimization problems for a class of natural queries called conjunctive queries. A conjunctive query can be expressed as a union of terminal conjunctive queries. We first characterized containment and minimization for terminal conjunctive queries. We then solved the problems of containment and optimization for the class of object-preserving conjunctive queries. The optimal queries are expressed as unions of terminal conjunctive queries. The notion of optimality captures the intuition that an optimal equivalent query logically accesses, in certain sense, the least number of objects in a database. It was shown that testing containment of terminal conjunctive queries is an

NP-complete problem. Moreover, the containment problem of conjunctive query in general is Π_2^p-complete.

Acknowledgement

The first author wishes to acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada. Work done in part while the first author was a visitor and the second author an employee of NTT Basic Research Labs, Tokyo. The authors also thank anonymous referees for their constructive comments which greatly enhanced the readability of 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] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. and Zdonik, S, "The Object-Oriented Database System Manifisto," *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. 40-57.

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

[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 3rd 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] *The Object Database Standard: ODMG-93*, Release 1.1, Cattell, R.G.G. (Editor), Morgan Kaufmann, San Mateo, CA, 1994.

[10] Chan, E.P.F., "Testing Satisfiability of a Class of Object-Oriented Conjunctive Queries," *Theoretical Computer Science (134), 1994*, pp. 287-309.

[11] Chan, E.P.F., "Containment and Minimization of Positive Conjunctive Queries in OODB's," *Proceedings of the 11th ACM Symposium on Principles of Database Systems*, San Diego, CA, 1992, pp. 202-211.

[12] 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.

[13] Cluet, S. and Delobel, C., "A General Framework for the Optimization of Object-oriented Queries," *Proceedings of the ACM SIGMOD*, San Diego, CA, 1992, pp. 383-392.

[14] 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.

[15] 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.

[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., Kim, W. and Sagiv, Y., "Querying Object Oriented Databases," *Proceedings of the 1992 ACM SIGMOD*, San Diego, CA, 1992, pp. 393-402.

[20] 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.

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

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

[23] Lanzelotte, R., Valduriez, P., Ziane, M. and Cheiney, J-P., "Optimization of Nonrecursive Queries in OODBs," *Proceedings of 2nd International Conference on Deductive and Object-Oriented Databases*, Munich, Germany, Lecture Notes in Computer Science 566, December 1991, pp. 1-21.

[24] Lecluse, C. and 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, CA, 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] Sagiv, Y. and Yannakakis, M., "Equivalences Among Relational Expressions with the Union and Difference Operators," *Journal of ACM 27(4)*, October 1980, pp. 633-655.

[29] Scholl, M.H., Laasch, C., and Tresch, M., "Updatable Views in Object-Oriented Databases," *Proceedings of 2nd International Conference on Deductive and Object-Oriented Databases*, Munich, Germany, Lecture Notes in Computer Science 566, December 1991, pp. 189-207.

[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] Straube, D.D. and Ozsu, T., "Queries and Query Processing in Object-Oriented Database Systems," *ACM Transactions on Information Systems 8(4)*, October 1990, pp. 387-430.

[32] StockMeyer, L.J., "The Polynomial-Time Hierarchy," *Theoretical Computer Science (3:1)*, 1976, pp.1-22.

[33] Stockmeyer, L.J. and Meyer, A.R., "Word Problems requiring exponential time," *Proc. of the 5th STOC*, pp.1-9, 1973.

[34] 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.

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

[36] Wrathall, C, "Complete Sets and the Polynomial-time Hierarchy," *Theoretical Computer Science (3)*, pp. 23-33, 1976.

[37] Zaniolo, C., "The Database Language GEM," *Proceedings of the 1983 ACM SIGMOD*, San Jose, CA, 1983, pp. 207-218.