Abstract Querying in a Tree-Structured Class Hierarchy

Chung-Dak Shum
Department of Computer Science
The HK University of Science and Technology
shum@cs.ust.hk

Richard R. Muntz
Computer Science Department
University of California, Los Angeles
muntz@cs.ucla.edu

Abstract

Class hierarchies are an important data modeling concept that forms the basis of most of today's object-oriented databases. We integrate the class hierachy concept in the formulation of queries as well as the presentation of answers. We formally define the notion of an abstraction in this context. The concept of an abstraction is applied in a graphical query language in which uninteresting classes are hidden. The answer to the query is also expressed as an abstraction. In a preferred abstract answer, classes subsumed by other class are not included. We show that the preferred abstract answer has the minimum number of classes and is unique in its representation. Labeled abstractions are introduced to accommodate exceptions in an abstract answer. A labeled abstraction is an abstraction whose classes are labeled. We consider expressing an abstract answer in terms of a restricted and then a general form of labeled abstraction. For either case, we consider the preferred answer as one that has the minimum number of classes in its representation. We show that the preferred answers are unique in their representations. Efficient algorithms for obtaining preferred answers follow from the constructive proofs. We extend the idea of an abstraction beyond a tree-structured class hierarchy to a partially ordered set (poset) of classes. In this more general context, preferred abstract answer under a poset of classes is no longer unique in its representation. Obtaining the preferred abstract answer under a poset of classes is shown to be an NP-complete problem. For multiple class hierarchies, which defines a restricted poset of classes, the problem remains intractable.

Index Terms - Answer representations, class hierarchies, graphical query languages, intensional answers, object-oriented databases.

1 Introduction

New data-intensive applications such as computer-aid design (CAD) [9], computer-aided software engineering (CASE) [11], multimedia databases [13], and geographical information systems (GIS) [6] pose new requirements that conventional databases oriented toward commercial applications fail to satisfy. These new applications require new data models, new query facilities and new transaction models. Among the requirements of these new applications are the support of complex objects, behavioral data, meta knowledge and long-duration transactions. Object-oriented databases are being developed to support these complex data-intensive applications. Issues in the design of object-oriented databases capable of supporting a rich collection of sophisticated data modeling concepts have become an important area of research [10,17]. Recent research efforts have also focused on providing facilities for querying object-oriented data models [1,2,14]. Most of these proposals for query processing in object-oriented databases are based on extending the relational framework to support various concepts of object orientation. Typically, the query facilities allow a user to retrieve a set of objects satisfying certain predicates and the answer to the query is given as an enumeration of individual object instances. The returned object instances may be interrelated. They may even belong to the same class in a class hierarchy. But none of this information is presented to the user nor do the query facilities allow the easy specification of such information. Class hierarchy is inarguably one of the more important concepts in object-oriented models. In this paper, we propose a graphical means to express queries as well as to present answers in the context of a class hierarchy.

Consider a personnel database of a small corporation and the query:

Q1: "Select all employees with a salary greater than 40K."

A conventional answer to the query is an enumeration of employee objects such as

A1: "John Smith, Peter Lee, James Daly, Allen Brown, Jim Blaine, Jerry Cox"

In the object-oriented model, similar objects are grouped naturally into classes and similar classes are in turn grouped to form super-classes. The recursive grouping of classes into super-classes forms a class hierarchy with each class as a specialization of its super-class. Figure 1 shows a simple example class hierar-

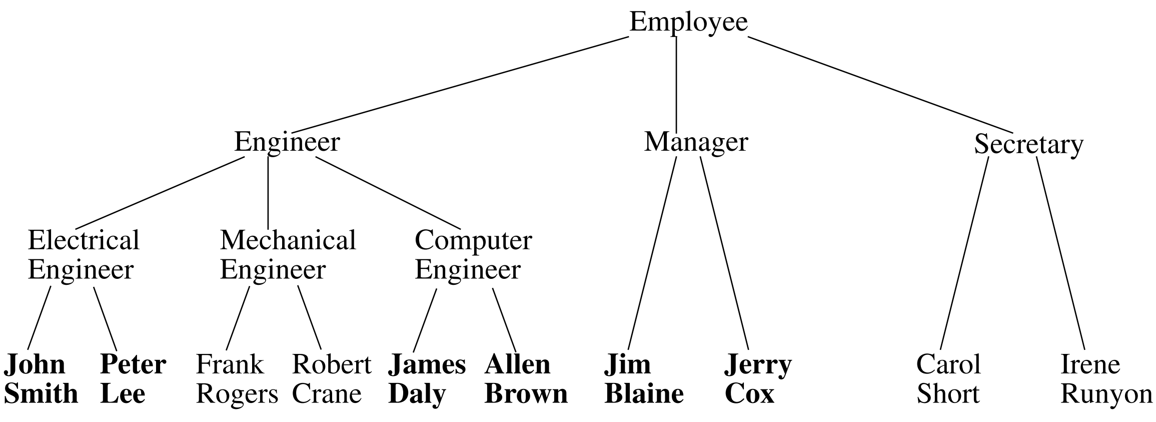

Figure 1: Employee Class Hierarchy

chy for our personnel database. The **boldface** employee objects correspond to the answers *A1* that satisfy the query *Q1*. It turns out that all *Electrical Engineers*, *Computer Engineers* and *Managers* satisfy the query. Graphically, in the presence of the employee class hierachy, the answers can also be presented as shown in Figure 2. Notice that the employee objects whose salaries are greater than 40K are not explicitly listed. We say that employee objects *John Smith* and *Peter Lee* are abstracted by the class *Electrical Engineer*. The ***hiding*** of objects (classes) or ***abstraction*** is the concept that we are going to exploit in expressing queries and in the presentation of answers. We term such an answer an ***abstract answer***. One of the advantages of presenting abstract answers is to facilitate information exchange at higher level of abstraction. When complex information is presented, the user may not be interested in too fine a detail. This is especially the case when the number of object instances satisfying the query is very large. Abstract answers can be very useful in decision support systems. It helps the user in acquiring some global understanding of the answer. Since explicit enumeration of object instances is not required, abstract answers also help in meeting resource constraints imposed by user interfaces.

The idea of an abstraction is not restricted only to the presentation of answers, but also applies to the querying for abstract answers. To obtain the abstract answer depicted in Figure 2, the query can be expressed graphically as shown in Figure 3. The meaning of the graphical query is similar to *Q1*; except that the answer is allowed to be expressed in terms of class concepts as well as object instances. Class concepts above the dotted line ***L*** are not shown or we say they are *hidden*. The answer is not allowed to contain class concepts such as *Engineer* or *Employee*. Thus the abstract answer given in Figure 2 is an answer to the graphical query. Another possible answer is shown in Figure 4. Notice that the former is expressed only in terms of class concepts whereas the latter is a combination of class concepts and object instances. Pro-

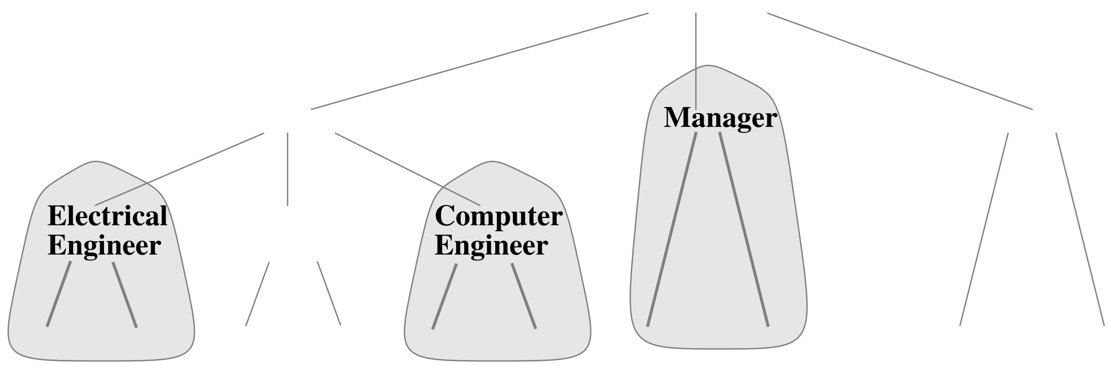

Figure 2: Example Abstract Answer.

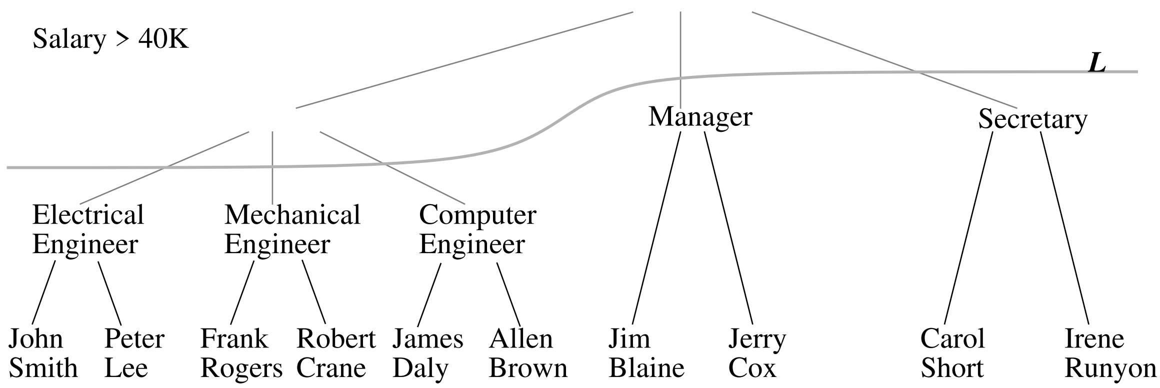

Figure 3: Example Graphical Query.

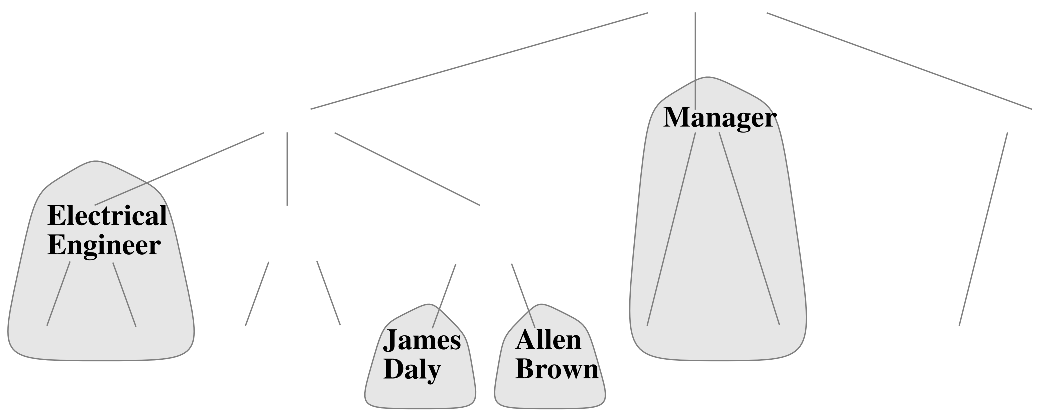

Figure 4: Another Possible Abstract Answer.

posing and evaluating criteria for the "goodness" of various abstract answers is another major contribution of this paper.

The remainder of this paper is organized as follows. Section 2 formally define the notion of an abstraction. In Section 3, we integrate the concept of an abstraction in the formulation of queries and the presentation of answers. In Section 4, we define the criteria for expressing preferences among abstract answers. We show that the preferred answer has the minimum number of classes and is unique in its representation. To accommodate exceptions in an abstract answer, Section 5 extends the concept of an abstraction to a labeled abstraction. We consider answers expressed in terms of a restricted as well as a general form of labeled abstraction. Under the same set of preference criteria, we show that the answer representations in both cases are unique in their representations and efficient algorithms exist in obtaining such answers. In Section 6, the idea of an abstraction is extended to a partially ordered set of classes. The representation of a preferred abstract answer is no longer unique. We also show that obtaining a preferred abstract answer is an NP-complete problem. In Section 7, we show that even for the case of multiple class hierarchies, a restricted poset of classes, the problem of obtaining a preferred abstract answer still remains intractable. Section 8 describes some work related to our study and how those work fit in our framework. Our conclusions and some suggestions of possible further work are given in Section 9.

2 Class Hierarchy and Abstraction

We consider a finite set **D** of object instances, and *classes* relative to **D**. A class is a unary predicate $C(\bullet)$ defined over D, where C, with possible subscripts, is the label of the class. For convenience, we will also denote the extension of the predicate $\{x \mid C(x)\}$ by C. The context should suffice to disambiguate. A class C_1 is said to be subsumed by an other class C_2 if and only if $C_1 \subseteq C_2$. We shall use both terminology (union, intersection, complementation, set inclusion, difference) and logic terminology (disjunction, conjunction, negation, subsumption) when referring to classes. We are not dealing with an arbitrary collection of classes; instead, we are interested in a class hierarchy.

Definition 1: A *class hierarchy* is a finite tree whose nodes are labeled by classes. Any node other than a leaf node has one or more successors. The successor classes of each are subsumed by their parent class. A class hierarchy is called *strict* if all sibling classes are mutually exclusive.

Definition 2: A *path* **P** in a class hierarchy **T** is a sequence of nodes in which successive nodes are connected by edges in **T**. A *rooted path* is any path whose first node is the root node. A *terminal path* is any path ending at a leaf node. A *full path* is a rooted terminal path.

Since we will be working mostly with strict class hierarchies, the term class hierarchy will refer to a strict class hierarchy unless otherwise stated. It is convenient to view an object instance as a *singleton* class of its own. Very often these singleton classes form the leaf nodes of a class hierarchy.

The number of classes in a class hierarchy can be very large and the user may not be interested in all classes or too fine a detail. For this reason, the hiding of classes or abstraction is an important concept in querying and the representation of answers in the presence of a class hierarchy. The hiding of a class from a class hierarchy changes the structure of a class hierarchy.

Definition 3: The *hiding* of a class from a class hierarchy results in one of the following structural changes:

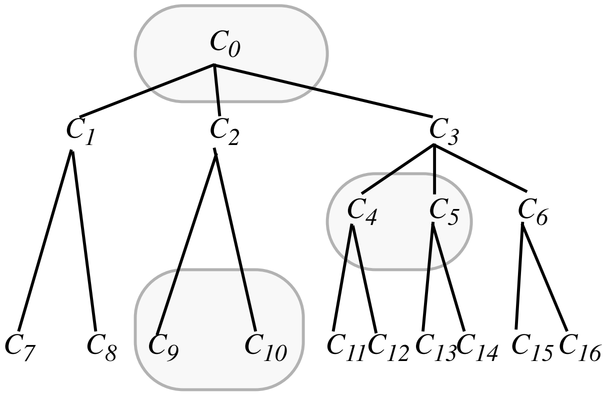

Figure 5a: A Class Hierarchy **T**

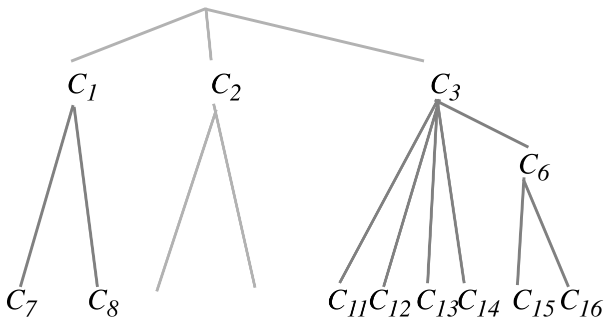

Figure 5b: An abstraction $\mathbf{T_A}$

1. If the class labels a leaf node, the leaf node is removed from the class hierarchy.
2. If the class labels a root node, each child becomes the root node of a class hierarchy.
3. If the class *C* labels any node other than a leaf or root node, each child of *C* becomes the direct child of *C*'s parent.

Definition 4: An *abstraction* $\mathbf{T_A}$ of a *class hierarchy* **T** is a forest of class hierarchies resulting from the hiding of one or more classes from **T**. An *abstraction* $\mathbf{T_A}$ of a *forest of class hierarchies* $\mathbf{F_T}$ is a forest of class hierachies resulting from the hiding of one or more classes from $\mathbf{F_T}$.

A graphical illustration of an abstraction is shown in Figure 5. A class hierarchy **T** is shown in Figure 5a. An abstraction $\mathbf{T_A}$ with hidden classes C_O, C_4, C_5, C_9 and C_{1O} is shown in Figure 5b. Notice that (i) the leaf nodes C_9 and C_{1O} are removed from the class hierarchy; (ii) with the removal of the root node C_O, $\mathbf{T_A}$ becomes three class hierarchies and *(iii)* C_3 becomes the parent of C_{11}, C_{12}, C_{13} and C_{14}. The meaning of an abstraction is related to its extension.

Definition 5: Let *class*($\mathbf{T_A}$) be the set of classes remaining in abstraction $\mathbf{T_A}$. The *extension* of an abstraction $\mathbf{T_A}$, denoted *extension*($\mathbf{T_A}$), is the set of object instances formed from $\bigcup_{C \in class(\mathbf{T_A})} C$. We also denote $|\mathbf{T_A}|$ as the number of classes in an abstraction $\mathbf{T_A}$.

Definition 6: Two abstractions $\mathbf{T_{A1}}$ and $\mathbf{T_{A2}}$ of a class hierarchy **T** are *equivalent* if and only if $extension(\mathbf{T_{A1}}) = extension(\mathbf{T_{A2}})$. They are *equal* if and only if $class(\mathbf{T_{A1}}) = class(\mathbf{T_{A2}})$.

Since an abstraction of a class hierarchy is characterized by its set of classes, we can infer a partial order between two abstractions.

Definition 7: Two abstractions $\mathbf{T_{A1}}$ and $\mathbf{T_{A2}}$ are related by the partial order $\mathbf{T_{A1}} \propto \mathbf{T_{A2}}$ if and only if $class(\mathbf{T_{A1}}) \subset class(\mathbf{T_{A2}})$.

3 Querying a Class Hierarchy

In relational databases, an object instance is represented as a tuple in a relation. Similar object instances in a relation can be thought of as forming a class. But there is no concept of class hierarchies. A construct for modeling class hierarchies does not exist in the relational model. Queries in relational databases typically ask for objects instances that satisfy certain conditions. Formally, a query can appear as

$$\{x \in D | P(x)\}$$

where D is the domain to which object instance x belongs and $P(x)$ is the condition that x has to satisfy. Most query languages proposed for object-oriented databases are based on extending the relational framework. Although the query languages allow the retrieval against classes, the target answer is still a set of object instances. The goal of a query language is to support the requests for information by a user and providing better facilities for a user to specify the presentation of an answer is clearly an advantage. Since the semantics of a class hierarchy are captured by the underlying object-oriented model, we advocate that those class hierarchical structures should also be incorporated in the querying and answering session.

Definition 8: A query over a class hierarchy **T** is defined as a 5-tuple $\langle \tilde{A}, \underset{\sim}{A}, \mathbf{T_A}, \mathbf{T}, P \rangle$. where P is the query condition, $\mathbf{T_A}$ is an abstraction of **T** and the abstract answer*s* $\tilde{A}$ *and* $\underset{\sim}{A}$ are abstractions of $\mathbf{T_A}$ such that

1. There exists no abstraction $\tilde{A}'$ such that $extension(\tilde{A}') \supset \{x \in extension(\mathbf{T_A}) | P(x)\}$ and $extension(\tilde{A}) \supset extension(\tilde{A}')$. We say that $extension(\tilde{A})$ minimally covers $\{x \in extension(\mathbf{T_A}) | P(x)\}$.
2. There exists no abstraction $\underset{\sim}{A}'$ such that $\{x \in extension(\mathbf{T_A}) | P(x)\} \supset extension(\underset{\sim}{A}')$ and $extension(\underset{\sim}{A}') \supset extension(\underset{\sim}{A})$. That is, $\{x \in extension(\mathbf{T_A}) | P(x)\}$ minimally covers $extension(\underset{\sim}{A})$

From Definition 8, it should be clear that $\mathbf{T_A}$ forms the basis of the answer to the query. Since the answer to the query is defined as an abstraction of $\mathbf{T_A}$, any classes belonging to the answer should also belong to $\mathbf{T_A}$. It is possible that the classes contained in the set *class*($\mathbf{T_A}$) are not detailed enough to exactly describe the extensional answer $\{x \in extension(\mathbf{T_A}) | P(x)\}$. For example, consider the class hierarchy **T** and its abstraction $\mathbf{T_A}$ from Figure 5a and b respectively. If an extensional answer to a query consists only of C_9, based on the abstraction $\mathbf{T_A}$, the exact answer cannot be expressed. Since the abstraction $\mathbf{T_A}$ is user specified, it contains the amount of detail that the user would like to see. Thus, a least upper bound $\tilde{A}$ and a greatest lower bound $\underset{\sim}{A}$ are defined as the answer to the query. This is similar to the definition of rough sets [12] which employs a lower and an upper approximation to define an “undefinable” set. The rational is that this is the maximal information expressible given the amount of detail the user is willing to see. From the

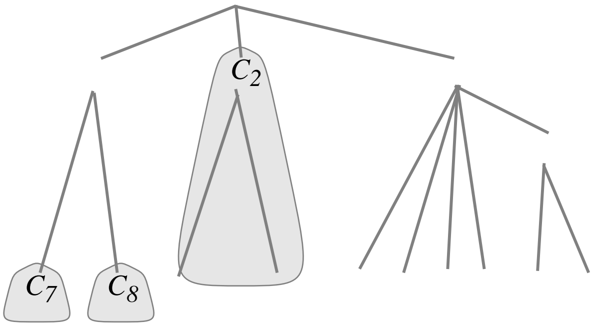

Figure 6a: Abstract Answer

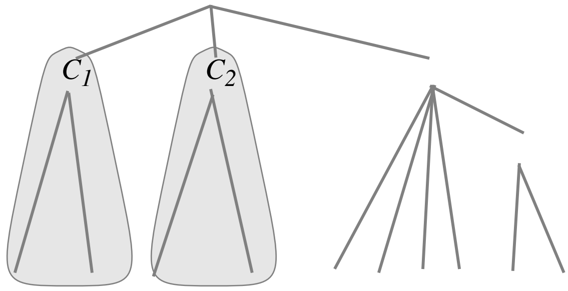

Figure 6b: Abstract Answer

definition of an abstraction, least upper bound and greatest lower bound answers are not necessary unique. This uniqueness issue will be further discussed in the next section.

4 Answer Representation

Our query specification (Definition 8) allows a user to specify the amount of detail he would like to see in an abstract answer. We have also shown that the amount of detail the user would like to see may not be enough to precisely express the exact answer. We provide the least upper and greatest lower bound answer to the query. It is obvious that if the exact answer can be expressed, a least upper bound answer is also the greatest lower bound answer or vice versa, but in general, neither a least upper bound answer nor a greatest lower bound answer is necessarily unique. Consider again the class hierarchy **T** and its abstraction $\mathbf{T_A}$ from Figure 5a and b respectively. Suppose C_7, C_8, C_9 and C_{10} are the only objects in *extension*($\mathbf{T_A}$) which satisfy the query predicate *P*. Then the abstract answers to the query $\langle \tilde{A}, \underset{\sim}{A}, \mathbf{T_A}, \mathbf{T}, P \rangle$ can be expressed exactly as shown in Figure 6a and b. It is easy to verify that either abstract answer can be the least upper bound $\tilde{A}$ and greatest lower bound $\underset{\sim}{A}$ to the query. The non-uniqueness of abstract answers allows a choice as to which answer we would prefer to see. Since one of our goals is to facilitate information exchange at higher level of abstraction, the abstract answer in Figure 6b is certainly more preferable. The idea of facilitating information exchange at higher level of abstraction applies equally to least upper bound answers $\tilde{A}$ as well as to greatest lower bound answers $\underset{\sim}{A}$. For simplicity, from this point on, we will only consider abstract answers that exactly express the answer. Extending the idea to least upper bound answers $\tilde{A}$ and greatest lower bound answers $\underset{\sim}{A}$ is straightforward.

Definition 9: Let A be an abstract answer and $C \in class(A)$. Then A is a *preferred* abstract answer if there exist no equivalent abstract answer A' such that $C' \in class(A')$ and C' is an ancestor of C.

The hiding of classes from a class hierarchy forms the basis of an abstraction. The hiding of more classes implies the suppression of more details. The number of classes appearing in an abstract answer is clearly one of the criteria against which to measure how "abstract" an answer is, in the sense of how much detail is hidden. The number of classes appearing in an abstract answer is related to the preferred abstract answer from Definition 8 in the following theorem.

Theorem 1: A preferred abstract answer A has the minimum number of classes in its representation.

Proof: Assume the contrary, then there exists an abstract answer A' with minimum number of classes such that $|A'| < |A|$. Since A and A' are equivalent, for any class $C \in class(A)$ and $C \notin class(A')$ there exists either (i) a class $C' \in class(A')$ where C' is an ancestor of C or (ii) a set of classes $\{C''_1, C''_2, \ldots, C''_k\} \subseteq class(A')$ where each C''_i is a descendant of C and $\bigcup_{1 \le i \le k} C''_i = C$. For the latter case (ii) where $k > 1$, A' will not have the minimum number of classes. The $k = 1$ case alone will not make $|A'| < |A|$. The former case (i) can make $|A'| < |A|$. But C' must have a set of descendant classes $\{C, C_1, \ldots, C_k\} \subseteq class(A')$ and $(\bigcup_{1 \le i \le k} C_i) \cup C = C'$ where $k \ge 1$. But by Definition 9, $\{C, C_1, \ldots, C_k\} \not\subset class(A')$. **Contradiction**. ■

The preferred abstract answer also has the property that it is unique.

Theorem 2: A preferred abstract answer A is unique in its representation.

Proof: Assume the contrary, then there exists another preferred abstract answer A' where for some $C \in class(A)$, $C \notin class(A')$. Since A and A' are equivalent, for any class $C \in class(A)$ and $C \notin class(A')$ there exists either (i) a class $C' \in class(A')$ where C' is an ancestor of C or (ii) a set of classes $\{C''_1, C''_2, \ldots, C''_k\} \subseteq class(A')$ where each C''_i is a descendant of C and $\bigcup_{1 \le i \le k} C''_i = C$. For case (i), by Definition 9, $C \notin class(A)$. For case (ii), again by Definition 9, for all i, $1 \le i \le k$, $C''_i \notin class(A')$. **Contradiction**. ■

The following property of a preferred abstract answer should also be obvious. The simple proof is omitted.

Proposition 1: Let A be a preferred abstract answer and $C \in class(A)$. If $C' \in class(\mathbf{T_A})$ and $C' \ne C$ is in the full path of C, then $C' \notin class(A)$.

A preferred abstract answer can be characterized by its classes. The algorithm for obtaining a preferred abstract answer is a simple postorder traversal of an abstraction. We assume the abstraction is a tree. Generalizing the algorithm to a forest is just a simple task of applying it to each tree in the forest.

Function **Preferred_Abstract_Answer**(**T_A**: abstraction, **P**: predicate): abstract_answer
Begin
class(**A**) := ∅;
Perform a postorder traversal of **T_A** and for each class **C** encountered *Do*
If **C** is a leaf node and **C** satisfies the predicate **P** *Then*
class(**A**) := class(**A**) + {**C**}
Else If all the children of **C** are in class(**A**) *Then*
Begin
Remove all children classes from class(**A**)
class(**A**) := class(**A**) + {**C**}
End
End postorder traversal
Return(A)
End

5 Extending the Expressive Power of an Abstraction

The preferred abstract answer is one way of representing an answer to a user. It facilitates information exchange at a high level since detail is suppressed. We associate the amount of detail with the number of classes in an abstract answer. The preferred abstract answer contains the least detail in the sense that it contains the minimum number of classes. We will also prove the nice property that a preferred abstract answer is unique. One of the drawbacks to the abstraction approach to representing answers is due to the fact that the classes in a class hierarchy often do not satisfy the query conditions as a whole. As a result, we cannot abstract answers to the extent we might want, except in a limited number of cases. For example, consider the class hierarchy **T** and its abstraction **T_A** from Figure 5a and b respectively. Suppose an extensional answer to a query consists of C_{11}, C_{12}, C_{13}, C_{14} and C_{15}. The preferred abstract answer is shown in Figure 7. Although the extensional answer "almost" equals the class C_3, we cannot use C_3 to abstract the answer because C_{16} is not part of the answer, but it is a descendant of C_3. If an *exception* is allowed in an

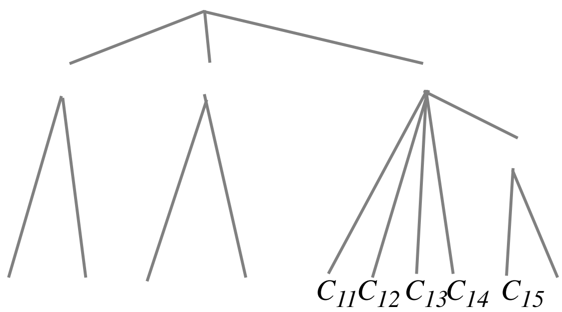

Figure 7: Preferred Abstract Answer

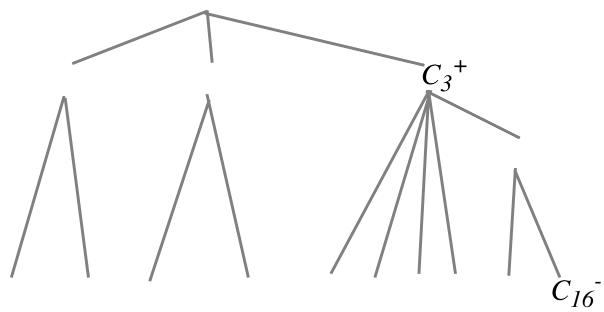

Figure 8: Abstract Answer with Exception

abstraction, an abstract answer can be represented as shown in Figure 8. The "-" denotes that the class is excluded. To be able to specify exceptions in an abstraction, we need to augment the definition of an abstraction.

Definition 10: A *labeled abstraction* is an abstraction whose classes are labeled by either a "+" or a "-".

The intuitive meaning of the label should be clear. When presented as an answer, a "+" label indicates that the class is in the answer; whereas, a "-" label indicates that the class is not part of the answer. Conflict occurs when the label of a class is different from the label of one of its descendants. In this case, the label for the subclass overrides the label for its superclass. This is in line with the concepts of inheritance and specialization in object-oriented modeling. In our case (Figure 8), a "+" label is associated with C_3 and a "-" label is associated with its descendant C_{16}. The answer "C_3 - C_{16}" is read as C_3 except C_{16}.

To formally capture and explain the meaning of a labeled abstraction, we introduce the notion of an expression.

Definition 11: The alphabet of an expression defined over an abstraction $\mathbf{T_A}$ of a class hierarchy $\mathbf{T}$ is composed of the following:

1. *Classes:* $C_1, C_2, \ldots,$
   Each class is a label of a node in $\mathbf{T_A}$.
2. *Empty:* ε,
   This denotes the empty expression.
3. *Signs:* +,-.

Next we introduce the notion of a literal, followed by the syntax of an expression.

Definition 12: There are two types of *terms*:

1. If C is a class, then $+C$ is a *positive* term.
2. If C is a class, then $-C$ is a *negative* term.

Definition 13: An *expression* over an abstraction $\mathbf{T_A}$ is defined recursively as follows:

1. A positive term is an expression.
2. ε is an expression.
3. If e is an expression and t is a term, then $e\ t$ is an expression.

Expressions are introduced as a convenient formalism for specifying a labeled abstraction. It is not difficult to see that the negative term is introduced to accommodate exceptions in an abstraction. Since we associate the meaning of an abstraction to the its extension, it is natural to associate the meaning of an expression to the set of object instances it represents.

Definition 14: The meaning of an expression over an abstraction $\mathbf{T_A}$ of a class hierarchy $\mathbf{T}$ is given by a mapping $\xi: Exp \rightarrow 2^R$, where *Exp* is the set of expression over $\mathbf{T_A}$ and *R* is the root class of $\mathbf{T}$.

1. $\xi(+C) = C$

$\xi(\varepsilon) = \varnothing$

2. $\xi(e+C) = \xi(e) \cup C$

$\xi(e-C) = \xi(e) - C$

Note that *C* is any class from the abstraction $\mathbf{T_A}$ and *e* is an expression over $\mathbf{T_A}$.

We assume that $\cup$ and - operators within a set expression have the same precedence and the left associative rule is used in the evaluation of such an expression. With the meaning of an expression defined, we can compare two expressions.

Definition 15: Two expressions e_1 and e_2 over an abstraction $\mathbf{T_A}$ are *equivalent* iff $\xi(e_1) = \xi(e_2)$.

Notice that by way of definition, expressions with same terms do not necessarily carry the same meaning. For example, if a class C_0 is the parent of class C_1 and class C_1 is the parent of class C_2 then the two expressions $e_1 = +C_0 - C_1 + C_2$ and $e_2 = +C_0 + C_2 - C_1$, although they contain the same terms, are not equivalent provided that $C_1 \neq C_2$. The relationship between an expression and a labeled abstraction is given by the following definition.

Definition 16: The pre-order traversal of a labeled abstraction $\mathbf{L_A}$ yields an *answer expression e* where

1. For each class C labeled "+" encountered in the traversal of $\mathbf{L_A}$, a positive term +*C* is appended to the expression *e*.
2. For each class C labeled "-" encountered in the traversal of $\mathbf{L_A}$, a negative term -*C* is appended to the expression *e*.

The meaning of a labeled abstraction is taken to be the meaning of its associated answer expression.

The motivation for introducing the concept of a labeled abstraction is to allow the further abstraction of an abstract answer which is otherwise not possible. The idea is still the hiding of classes; more precisely, the hiding of labeled classes. The number of labeled classes appearing in an answer is again one of the criteria which we use to measure how abstract an answer is. From Definition 16, it is clear that each labeled abstraction yields a unique answer expression. Conversely, each answer expression corresponds to one labeled abstraction. Thus, one criterion for a good answer can be translated to finding an answer expression with the minimum number of terms.

Introducing exceptions in an abstract answer allows us to further abstract an answer. But unless it helps in reducing the number of labeled classes in a labeled abstraction, its presence may not be very desirable. In querying, it is assumed that the user is interested to know which classes satisfied the query condition, not the classes that do not satisfy the query condition. A second criterion, after satisfying the minimum number of terms, is to minimize the number negative terms.

Preferred abstract answers are the simplest possible. In a preferred abstract answer, as illustrated in Proposition 1, there is at most one class from each full path. This is, however, no longer valid with the introduction of labeled abstractions. When expressing exceptions in a labeled abstraction, the answer may include more than one (labeled) class from the same full path. For example, in Figure 8, the example answer has two classes, C_3 and C_{16}, and they are both from the same full path. To retain some simplicity, it is possible to introduce further restriction on the answer expressions.

> **Definition 17:** A *restricted answer expression* is an answer expression over an abstraction $\mathbf{T_A}$ such that there exists at most one positive term and one negative term whose corresponding classes belong to the same full path in $\mathbf{T_A}$.

The definition of a restricted answer expression is influenced by the property stated in Proposition 1. Figure 9 illustrates a graphical representation of a restricted answer expression. A restricted answer expression corresponds to a forest of trees in which each tree has a root node with a "+" label and zero or more descendants with "-" labels and all other nodes are hidden. Apparently, such representations are simple and easy to comprehend. Figure 10 shows a graphical representation when a general answer expression is used

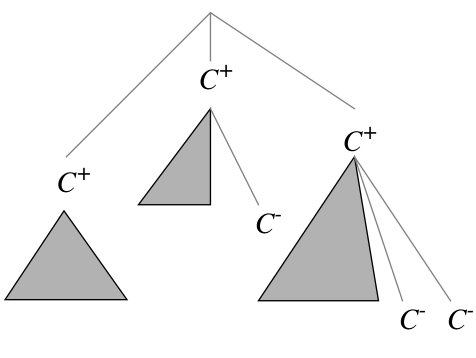

Figure 9: Restricted Answer Expression

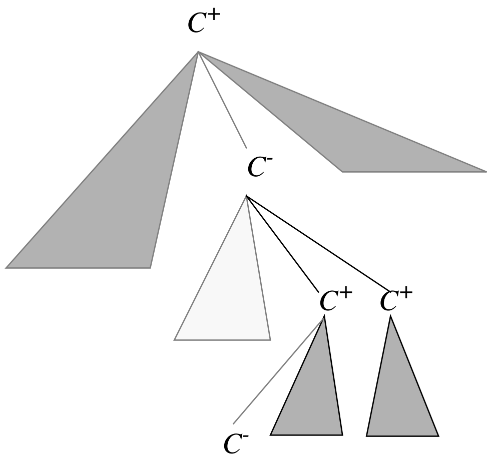

Figure 10: General Answer Expression

to represent an abstract answer. A general answer expression also corresponds to a forest of trees, although Figure 10 only shows one tree. Each tree can be complicated and has a number of levels. But the abstract response still provides the user with some high level information regarding the classes of objects that satisfy the query. In the next two sections, we will study the representation of abstract answers using restricted answer expressions and general answer expressions.

5.1 Restricted Answer Expression

It will be shown in this section that restricted answer expression can be used to represent abstract answers. There are two criteria for comparing restricted answer expressions. The first criterion is the number of terms in an expression. The fewer the number of terms, the more abstract the answer is. When the number of terms in two expressions are the same, the one with the fewer number of negative terms is preferred.

Definition 18: Given a set of equivalent restricted answer expressions over an abstraction $\mathbf{T_A}$. A restricted answer expression $\hat{e}$ is *preferred* if it satisfies the following properties.

1. $\hat{e}$ has a minimum number of terms.
2. If an equivalent expression e has the same number of terms as $\hat{e}$, $\hat{e}$ has a fewer number of negative terms.
3. If a class C is the only child of a parent C', $\hat{e}$ does not contain either the term $+C$ or the term $-C$.

Property 3 is consistent with the spirit of Definition 9 for preferred abstract answer. Abstract answers are expressed in terms of classes in the highest possible level in a class hierarchy. From the definition, a preferred restricted answer expression has the minimum number of terms. In the following, we will show that a preferred restricted answer expression is also unique. First, we define what we mean by a subexpression.

Definition 19: Let e be a restricted answer expression over an abstraction $\mathbf{T_A}$ and $\mathbf{T_A}^C$ be a subtree of $\mathbf{T_A}$ rooted at class C. A *subexpression* of e over a class C is e, with every term $\pm C'$ in e whose corresponding class C' is not in $\mathbf{T_A}^C$, removed.

A subexpression is a restricted answer expression which only concerns a subtree from $\mathbf{T_A}$.

Theorem 3: A preferred restricted answer expression $\hat{e}$ is unique in its representation.

Proof: Assume the contrary, then there exists another preferred restricted answer expression $\hat{e}'$ such that $\hat{e}'$ and $\hat{e}$ has the same number of positive as well as negative terms, but

Case (I): $\hat{e}'$ has some term $+C'$ not in $\hat{e}$.

Let us consider the meaning represented by the subexpression $e_{(0)}$ of $\hat{e}'$ over the class C'. Since $\hat{e}$ and $\hat{e}'$ are equivalent, there must exist either (i) a subexpression $e_{(i)} = +\underset{\sim}{C}_{10} - \underset{\sim}{C}_{11} - \underset{\sim}{C}_{12} \ldots - \underset{\sim}{C}_{1k_1} + \underset{\sim}{C}_{20} - \underset{\sim}{C}_{21} - \underset{\sim}{C}_{22} \ldots - \underset{\sim}{C}_{2k_2} \ldots + \underset{\sim}{C}_{j0} - \underset{\sim}{C}_{j1} - \underset{\sim}{C}_{j2} \ldots - \underset{\sim}{C}_{jk_j}$ of $\hat{e}$ over the class C' where all $\underset{\sim}{C}_{xy}$ are descendants of C' or (ii) a subexpression $e_{(ii)} = +C - C_1 - C_2 \ldots$ of $\hat{e}$ over C where C is an ancestor of C'.

Subcase (i): The number of terms in $e_{(i)}$ is $j + (k_1 + k_2 + \ldots + k_k)$,where the summation of the k_i represents the number of negative terms. Each $-\underset{\sim}{C}_{xy}$ term in subexpression $e_{(i)}$ indicates that the object instances in class $\underset{\sim}{C}_{xy}$ do not belong to the answer. Since $\underset{\sim}{C}_{xy}$ is a descendant of C', $-\underset{\sim}{C}_{xy}$ must also belong to $e_{(0)}$. If $\hat{e}'$ is a preferred restricted answer expression, $e_{(0)}$ also has $1 + (k_1 + k_2 + \ldots + k_k) + (j - 1)$ terms. The "1" accounts for the term $+C'$, the summation of k_i represents the number of negative terms that also belong to $e_{(i)}$ and the last "j - 1" accounts for other negative terms. If $j \neq 1$, $e_{(i)}$ is preferred since it has fewer number of negative terms. If $j = 1$, by Property 3 of Definition 18, again $e_{(i)}$ is preferred. Thus $\hat{e}'$ is not a preferred restricted answer expression. **Contradiction**.

Subcase (ii): If there is no C'' under C such that $C'' \cap C = \varnothing$, by Property 3 of Definition 18, $e_{(ii)}$ is preferred. Otherwise, consider the subexpression $e_{(iii)}$ of $\hat{e}'$ over the class C. Each negative term in subexpression $e_{(iii)}$ indicates that the object instances in the corresponding class do not belong to the answer. Since C is an ancestor of all those classes, every negative term must also belong to $e_{(ii)}$. Thus, $e_{(ii)}$ has at least as many negative terms as $e_{(iii)}$. If $e_{(ii)}$ has more negative terms, then $\hat{e}$ is not a preferred restricted answer expression. Otherwise, $e_{(ii)}$ has less terms than $e_{(iii)}$. $\hat{e}'$ is not a preferred restricted answer expression. **Contradiction**.

Case (II): $\hat{e}'$ has some term $-C'$ not in $\hat{e}$.

Since $\hat{e}'$ has the minimum number of terms, $-C'$ will not be in $\hat{e}'$ unless there exists another term $+C$ in $\hat{e}'$ where C is an ancestor of C'. From Case (I), $+C$ must also be in $\hat{e}$. For $\hat{e}$ and $\hat{e}'$ to have the same meaning, the object instances represented by C' must be excluded from $\hat{e}$. Without the term $-C'$, there must exist terms $-C_1' - C_2' \ldots - C_n'$ in $\hat{e}$ such that C_j''s are descendants of $-C'$ and by Definition 18, $n > 1$. Thus, $\hat{e}$ does not have the minimum number of terms and $\hat{e}$ is not a preferred restricted answer expression. **Contradiction**. ■

The algorithm for obtaining a preferred restricted answer expression is a recursive function. We also make use of the Preferred_Abstract_Answer function defined in Section 4. Since the output of the function is an abstraction, for convenience, we redefine the function as **Preferred_Abstract_Expression**. The output of this function is an expression constructed from a pre-order traversal of the original output abstraction. For each class C encountered in the abstraction, a *negative* term $-C$ is appended to the expression. We generate negative terms because we only use this function to identify classes not belonging to an answer. Again we assume the abstraction is a tree. Generalizing the algorithm to a forest is just a simple matter of applying it to each tree in the forest.

Function **Preferred_Restrict_Expression**($\mathbf{T_A}$: abstraction, **P**: predicate): expression
Begin
Let **R** be the root class of $\mathbf{T_A}$ and $\mathbf{T_A^1}$, $\mathbf{T_A^2}$, ... ,$\mathbf{T_A^k}$ be **k** subtrees under **R**.

/* Note: listing two expressions one after another defines an append operation */
Let α = **Preferred_Restrict_Expression**($\mathbf{T_A^1}$, **P**) ...
Preferred_Restrict_Expression($\mathbf{T_A^k}$, **P**)
Let β = **+R Preferred_Abstract_Expression**($\mathbf{T_A^1}$, $\neg$**P**) ...
Preferred_Abstract_Expression($\mathbf{T_A^k}$, $\neg$**P**)

If the number of terms in α and β are different *Then*
Return the one with fewer terms
Else
Return the one with fewer negative terms
End

5.2 General Answer Expression

This section deals with the representation of abstract answers using general answer expressions. The criteria for comparing answer expressions are the same as those for comparing restricted answer expressions.

Definition 20: Given a set of equivalent answer expressions over an abstraction $\mathbf{T_A}$. An answer expression $\hat{e}$ is *preferred* if it satisfies the following properties.

1. $\hat{e}$ has the minimum number of terms.
2. If an equivalent expression *e* has the same number of terms as $\hat{e}$, $\hat{e}$ has a fewer number of negative terms.
3. If a class C is the only child of a parent C', $\hat{e}$ does not contain the term $+C$ or $-C$.

From the definition, a preferred answer expression has the minimum number of terms. Since a restricted answer expression is an answer expression, a preferred answer expression must use at most the same number of terms that a preferred restricted answer expression uses in expressing the same abstract answer. In the following, we will show that a preferred answer expression is unique. First, we define a path subexpression and prove a lemma that will be useful in establishing the uniqueness theorem.

Definition 21: Let e be an answer expression over an abstraction $\mathbf{T_A}$ and $\mathbf{P}$ be a full path in $\mathbf{T_A}$. A *path subexpression* of e over $\mathbf{P}$ is an expression formed by removing from e all those terms whose associated class are not in $\mathbf{P}$.

Lemma 1: Let $\hat{e}$ be a preferred answer expression. Then any path subexpression $\hat{e}_P = \pm C_{P_1} \pm C_{P_2} \ldots \pm C_{P_k}$ of $\hat{e}$ over some full path $\mathbf{P}$ must have alternate positive and negative terms.

Proof: Assume the contrary. There are two cases:

(i) $\hat{e} = e_H + C_{P_i} e_{notP} + C_{P_{i+1}} e_T$

where e_H and e_T represent respectively the head and tail portion of $\hat{e}$ and e_{notP} represents the portion between $+C_{P_i}$ and $+C_{P_{i+1}}$.

Note that, by the definition of subexpression of $\hat{e}$ over $\mathbf{P}$, all classes associated with e_{notP} are not in $\mathbf{P}$, and $C_{P_{i+1}} \subseteq C_{P_i}$. Thus, object instances in $C_{P_{i+1}}$, being introduced by the term $+C_{P_i}$, are not affected by e_{notP}. So the term $+C_{P_{i+1}}$ is redundant.

Therefore, $\hat{e}$ is not a preferred answer expression. **Contradiction**.

(ii) $\hat{e} = e_H - C_{P_i} e_{notP} - C_{P_{i+1}} e_T$

The proof of this case is similar to (i). ■

Now we are ready to establish that a preferred answer expression is unique. We use the term *preferred expression* to refer to those expressions (not necessarily answer expressions) that satisfy the above three criteria given in Definition 20. For simplicity, we also assume that the abstraction is a tree and each non-leaf class has at least two child classes. Extending the proof to a general forest of tree is a simple extension.

Theorem 4: A preferred answer expression $\hat{e}$ is unique in its representation.

Proof: Since an answer expression is a pre-order traversal of a labeled abstraction, for a preferred answer expression $\hat{e}$ to be unique in its representation, we only need to show the set of terms making up $\hat{e}$ is unique. The proof proceeds by induction on the height of the tree abstraction. The inductive proof also elicits an algorithm for obtaining a preferred answer expression.

$height = 1$ (a root class R with leaf classes C_i attached)

Consider two ways of expressing the answer:

(i) without root class R: $^R = \begin{cases} \varepsilon & \text{if the answer is null} \\ +C_{i_1} \ldots + C_{i_k} & \end{cases}$

(ii) with root class R: $e_1^{+R} = +Re_1^-$ where

$$e_1^- = \begin{cases} \varepsilon & \text{if all children of R belong to the answer} \\ -R & \text{if answer is null} \\ -C_{j_1} \ldots - C_{j_m} & m < \dfrac{|R| + 1}{2} \\ -R + C_{i_1} \ldots + C_{i_k} & \text{otherwise} \end{cases}$$

We claim that

(1) e_1^{-R} is a preferred expression conditioned on R being excluded;

(2) e_1^{+R} is a preferred expression conditioned on R being included;

(3) except for the case where all children of R belong to the answer, e_1^{+R} has more negative terms than e_1^{-R}.

From (1), (2) and (3), we conclude that only one of e_1^{+R} or e_1^{-R} will contain the set of terms making up the preferred answer expression (e_1^{pre}).

In the inductive proof, we will establish properties (1), (2) and (3) for trees of arbitrary height n and conclude that only one of e_n^{+R} or e_n^{-R} will contain the set of terms making up the preferred answer expression (e_n^{pre}).

$height = k$

Assume that

(1) e_k^{-R} is a preferred expression conditioned on the root class being excluded;

(2) $e_k^{+R} = +Re_k^-$ is a preferred expression conditioned on the root class being included;

(3) except for the case where all children of R belong to the answer, e_k^{+R} has more negative terms than e_k^{-R}.

From (1), (2) and (3), we conclude that one of e_k^{+R} or e_k^{-R} will contain the set of terms making up the preferred answer expression e_k^{pre}.

$height = k+1$

Assume that the root R has n children $(S_1, \ldots S_n)$ and each child is the root of a tree with height k. For each subtree T_i rooted at child S_i $(1 < i \leq n)$, $e_{k_i}^{-R}$ and $e_{k_i}^{+R} = S_i e_{k_i}^-$ exist and have the properties as described above $(height = k)$.

Again consider two ways of expressing the answer:

(i) without root class R: $e_{k+1}^{-R} = e_{k_1}^{pre} \ldots e_{k_n}^{pre}$

It is easy to see that

(1) e^{-R}_{k+1} *is a preferred expression conditioned on the root concept being excluded.*

Without the root, the subtrees T_i are mutually independent. Thus the only "preferred" way to express the answer is to concatenate preferred expressions from each subtree.

(ii) with root class R: $e^{+R}_{k+1} = +Re^{-}_{k+1}$ where

$$e^{-}_{k+1} = \begin{cases} \varepsilon & \text{if all children of R belong to the answer} \\ -R & \text{if answer is null} \\ PRE\,(e^{-}_{k_1}\dots e^{-}_{k_n}, -Re^{pre}_{k_1}\dots e^{pre}_{k_n}) & \text{otherwise} \end{cases}$$

The *PRE* function simply returns the argument expression which is the preferred of the two.

We note that expression $(\alpha)\, e^{-}_{k_1}\dots e^{-}_{k_n}$ and $(\beta) -Re^{pre}_{k_1}\dots e^{pre}_{k_n}$ cannot have the same number of positive and negative terms.

To show that, we compare each pair of terms $e^{-}_{k_i}$ and $e^{pre}_{k_i}$ in α and β respectively. Let $|\bullet|$ denotes the number of terms in an expression. Consider the following cases:

1. if $\left|e^{-R}_{k_i}\right| > \left|e^{+R}_{k_i}\right|$ then
   $e^{pre}_{k_i} = e^{+R}_{k_i} = S_i e^{-}_{k_i}$ and $e^{pre}_{k_i}$ has an extra *positive* term over $e^{-}_{k_i}$.
2. if $\left|e^{-R}_{k_i}\right| = \left|e^{+R}_{k_i}\right|$ then
   $e^{pre}_{k_i} = e^{-R}_{k_i}$ and $e^{pre}_{k_i}$ has one more term but less *negative* term than $e^{-}_{k_i}$.
3. if $\left|e^{-R}_{k_i}\right| + 1 = \left|e^{+R}_{k_i}\right|$ then
   $e^{pre}_{k_i} = e^{-R}_{k_i}$ and $e^{pre}_{k_i}$ has less *negative* term than $e^{-}_{k_i}$.
4. otherwise $\left|e^{-R}_{k_i}\right| + 2 = \left|e^{+R}_{k_i}\right|$ and thus
   $e^{pre}_{k_i} = e^{-R}_{k_i}$ and $e^{-}_{k_i} = -S_i e^{-R}_{k_i}$ and $e^{-}_{k_i}$ has an extra *negative* term.

Assume that α and β can have the same number of positive and negative terms and there are *w* case 1, *x* case 2, and *y* case 3, and *z* case 4. If α and β have the same number of terms,

$$w + x + 1 = z \qquad (I)$$

The $+1$ comes from $-R$ in β. If the expressions have the same number of negative terms,

$$x + y + z \le 1 \qquad (II)$$

Again the 1 on the right hand side of the inequality comes from $-R$ in β. Furthermore, the number of successors of R is at least two, that is,

$$w + x + y + z \ge 2 \qquad (III)$$

Since *w*, *x*, *y*, and *z* are all natural numbers, it should be obvious that no solution can satisfy (I), (II) and (III) simultaneously. **Contradiction**. Thus, the expressions α and β cannot have the same number of positive and negative terms.

Next we show that

(2) e^{+R}_{k+1} *is a preferred expression conditioned on the root class being included.*

The case where all the children of R belong to the answer and the case where the answer is null are trivial. We only show the last case. Suppose $e^{+R}_{k+1} = +Re^{-}_{k_1}\dots e^{-}_{k_n}$.

Assume that there exists $e^{+R}_{k+1}{}' = +Re^{-}_{k_1}\dots e^{-}_{k_i}{}'\dots e^{-}_{k_n}$ which is preferred conditioned on R being included. By Lemma 1, for any full path, any class in $e^{-}_{k_i}$ or $e^{-}_{k_i}{}'$ which is closest to R must be negative.

Thus for subtree T_i we can get $+S_i e^{-}_{k_i}{}'$ which is preferred conditioned on S_i being included instead of $e^{+R}_{k_i}$. **Contradiction**.

Similarly, we can show the same for $e^{+R}_{k+1} = +R - Re^{pre}_{k_1}\dots e^{pre}_{k_n}$.

Also

(3) e^{+R}_{k+1} *has more negative terms than* e^{-R}_{k+1}.

This is true since each $e^{+R}_{k_i}$ has more negative terms than $e^{-R}_{k_i}$.

Hence, from (1), (2) and (3) one of e^{+R}_{k+1} or e^{-R}_{k+1} must be a preferred expression and the set of terms making up the preferred answer expression must be unique. ■

As mentioned, the proof of Theorem 4 is constructive in nature, an algorithm for obtaining a preferred answer expression over an abstraction $\mathbf{T_A}$ falls naturally out of its induction step. Basically, we do a postorder traversal of $\mathbf{T_A}$, that is, before a node is visited, all its descendants must have been traversed. Expressions $e^{+R}_{k_i}$ and $e^{-R}_{k_i}$ are constructed as each node (indexed by k_i) is being encountered and the preferred answer expression is readily obtainable once the root has been visited.

6 Partially Ordered Classes

A collection of classes relative to a domain **D** does not necessarily fall into a strict (tree) hierarchy. In other words, for two classes, it may neither be the case that their intersection is empty nor one is the subset of the other. For example, consider two job categories of the employees of a company: *computer scientist* and *electrical engineer.* It is not surprising that some employees belong to both categories, whereas others belong exclusively to one category. In fact, if enough members belong to the intersection, it is quite natural to form another class, say, *computer engineer*, so that we can denotate them collectively as an aggregate. A simple extension beyond the strict hierarchy leads us to consider a *partially ordered set* (*poset*) of classes.

A graphical illustration of a poset P is shown in Figure 11. The classes C_1 and C_2 in P are related by the partially ordered relation $C_2 \le C_1$ and the relation is shown graphically by the edge connecting C_1 and C_2 from top to bottom. C_1 and C_7 are also related by the partially ordered relation $C_7 \le C_1$. But the relation is implied by transitivity and not explicitly shown. In this case $C_2 \le C_1$ and $C_7 \le C_2$ imply $C_7 \le C_1$. The idea of hiding classes and the definition of an abstraction of a class hierarchy extend naturally to a poset of classes. Figure 11b provides an illustration of an abstraction of P. It should also be apparent that, in our case, the partially ordered relation is taken as the *subset* relation. That is, for $C_1, C_2 \in P$, $C_2 \le C_1$ if and only if $C_2 \subseteq C_1$.

Let us consider, instead of querying a class hierarchy, querying a partially ordered set of classes. The definitions applying to a class hierarchy in Section 2, 3 and 4 can be generalized to apply to a partially ordered set of classes. The generalizations should be trivial and we will not restate the definitions here. It would be ideal if after the generalization to a poset of classes, the preferred abstract answer still has the minimum number of classes and is unique in its representation. But a simple example shows that this is no longer the case. Suppose, in Figure 11a, the extensional answers to the query consists only of C_6, C_7 and C_8. Then the preferred abstract answer to the query can be expressed either as shown in Figure 12a or Figure 12b. Thus, Theorem 2 is not always true under a poset of classes.

The hiding of details is still our main concern. Without worrying about the uniqueness in representation, we would like to know whether a preferred abstract answer, one that has a minimum number of

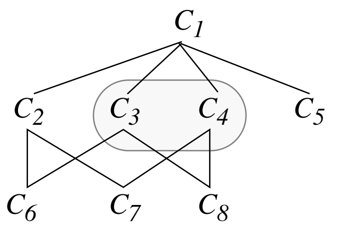

Figure 11a: A Poset P of Classes

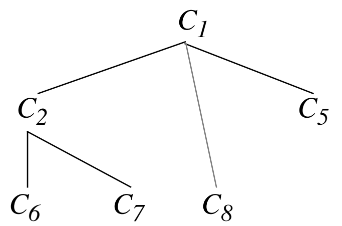

Figure 11b: An Abstraction P

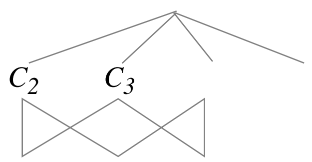

Figure 12a: Abstract Answer

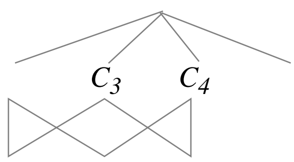

Figure 12b: Abstract Answer

classes in its representation, can be obtained through an efficient algorithm. Unfortunately, we are able to show that the problem at hand belongs to the class of NP-complete problems. First, let us restate our problem in terms of a decision problem.

Minimal Abstract Answer

INSTANCE: An abstraction $\mathbf{P_A}$ of a poset $\mathbf{P}$ of classes, a set of classes $\mathbf{Q}$ where $\mathbf{Q} \subseteq class(\mathbf{P_A})$ and each class in $\mathbf{Q}$ satisfies the query, and a bound $\mathbf{B} \in Z^+$ (where Z^+ denotes the positive integers).

QUESTION: Is there an abstraction $\mathbf{A}$ of $\mathbf{P_A}$ such that $extension(\mathbf{A}) = \bigcup \mathbf{Q}$ and the number of classes in $\mathbf{A}$ is no more than $\mathbf{B}$?

Clearly, if we could find an abstraction with the minimum number of classes for describing $\mathbf{Q}$ in polynomial time, we could also solve the associated decision problem described above in polynomial time. All we need to do is find such abstraction, determined the number of classes involved, and compare it with the given bound $\mathbf{B}$. Thus, the decision problem can be no harder than the corresponding optimization problem. If we could demonstrate the Minimal Abstract Answer problem is NP-complete, we would know that its optimization problem is at least as hard.

Theorem 5: Minimal Abstract Answer is NP-complete.

Proof: It is easy to see that Minimal Abstract Answer $\in$ NP since a non-deterministic algorithm need only guess an abstraction $\mathbf{A}$ and check in polynomial time whether $extension(\mathbf{A}) = \bigcup \mathbf{Q}$ and has the appropriate number of classes.

The remaining part of the proof involves the transformation of some already know NP-complete problem to our Minimal Abstract Answer problem and showing that the transformation can be done in polynomial time. We transform Vertex Cover to Minimal Abstract Answer.

Vertex Cover

INSTANCE: A graph $G = (V,E)$ and a positive integer $K \leq |V|$.

QUESTION: Is there a *vertex cover* of size K or less for G, that is, a subset $V' \subseteq V$ such that $|V'| \leq K$ and, for each edge $\{u, v\} \in E$, at least one of u and v belongs to V?

The technique we use in the transformation is local replacement. Let an arbitrary instance of Vertex Cover be given by the graph $G = (V,E)$ and the positive integer $K \leq |V|$. We must construct a set of classes $\mathbf{Q} \subseteq class(\mathbf{P_A})$ such that G has a vertex cover of size K or less if and only if an abstraction $\mathbf{A}$ of $\mathbf{P_A}$ satisfying $extension(\mathbf{A}) = \bigcup \mathbf{Q}$ contains no more than $\mathbf{B}$ classes.

The local replacement just substitutes for each edge $\{u, v\} \in E$ a class $\mathbf{uv} \in \mathbf{Q}$ and for each vertex $v \in V$ a class $\mathbf{v}$. Next, we introduce an ordering for the defined classes as follows:

$\mathbf{uv} \leq \mathbf{v}$ iff v is a vertex of the edge $\{u, v\}$

Figure 13 shows an example of $\mathbf{P_A}$ obtained from a graph G.

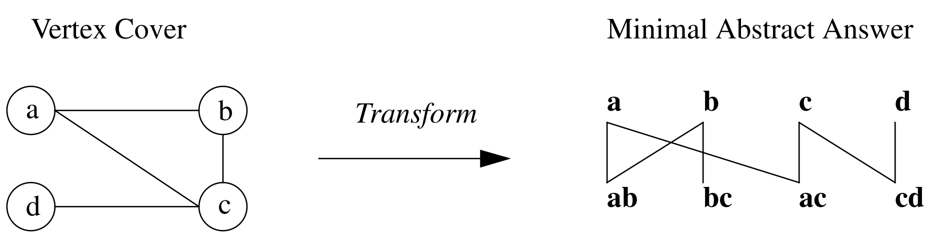

Figure 13: Minimal Abstract Answer is NP-Complete

It is obvious that the construction can be accomplished in polynomial time. All that remains to be shown is that G has a vertex cover of size K or less if and only if an abstraction $\mathbf{A}$ of $\mathbf{P_A}$ satisfying $extension(\mathbf{A}) = \bigcup \mathbf{Q}$ has no more than $\mathbf{B} = K$ classes. This should be apparent by way of construction. ∎

7 Multiple Class Hierarchies

In object-oriented model, similar objects are grouped naturally into classes and the recursive grouping of classes forms a class hierarchy. Since there can be more than one way to group objects, objects can belong to more than one class, each such class belonging to a different class hierarchy. In Figure 1, the employee objects belong to a *job category* class hierarchy. The employee objects can also belong to another *education level* class hierarchy. Consider again the query

Q1: “Select all employees with a salary greater than 40K.”

Usually, an employee’s salary is related to his job. Also, it may be the case that his salary is related to his education. It would be convenient if in addition to querying a single class hierarchy, we can query a *composition* of multiple class hierarchies. An example answer to the above query is

A1: “All manager and all engineers with master degrees.”

Notice that there are three classes in the answer. Of the three classes involved, *Manager* and *Engineer* belong to the *job category* class hierarchy, whereas *Master* belongs to the *education level* class hierarchy. In this Section, we are concerned with querying a composition of multiple class hierarchies.

To express the set of *“all engineers with master degrees”*, we introduce the notion of a *composite class.*

Definition 22: Let $\mathbf{T_i}$ $(1 \le i \le k)$ be a set of class hierarchies and C_i be a class of $\mathbf{T_i}$. Then the tuple $\langle C_1 \ldots, C_k \rangle$ is a *composite* class over k class hierarchies. We also referred to each C_i as a simple class.

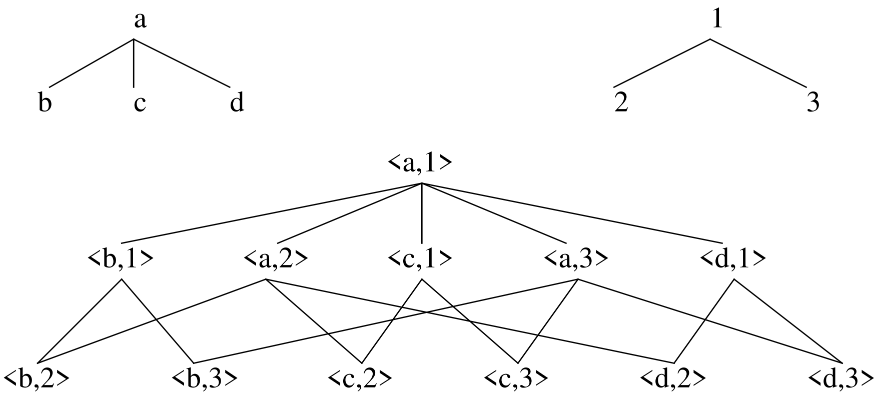

Figure 14: A Direct Product of two Class Hierarchies

If C_1 of $\mathbf{T_1}$ denotes the set of *engineers* and C_2 of $\mathbf{T_2}$ denotes the set of *masters*, we use the composite class $\langle C_1, C_2 \rangle$ to denote the set of *all engineers with master degrees*. As is true for all classes, a composite class represents a set of objects; but, instead of being defined inherently as in simple classes, it is given in terms other simple classes.

> **Definition 23:** Let $\langle C_1 \ldots, C_k \rangle$ be a composite class over *k* class hierarchies. The extension of $\langle C_1 \ldots, C_k \rangle$ is defined as $\bigcap_{1 \le i \le k} C_i$.

The collection of composite concepts that we are dealing with is not arbitrary. Their relationships with one another can be derived from the following definition.

> **Definition 24:** The *direct product* $T_1 \ldots T_k$ of *k* class hierarchies T_i, $1 \le i \le k$, is the set of all tuples $\langle x_1, \ldots, x_k \rangle$ with $x_i \in P_i$, partially ordered by the rule that $\langle x_1, \ldots x_k \rangle \le \langle y_1, \ldots, y_k \rangle$ if and only if $x_i \le_{Ti} y_i$ for all $i \; (1 \le i \le k)$.

If the partial order relations are taken as the subsumption relations, it is not difficult to see that our definition of composite classes satisfies the ordering introduced by Definition 24 and our collection of composite classes over *k* classes hierarchies corresponds to the direct product of *k* class hierarchies. Figure 14 shows an example of a direct product of two class hierarchies.

From Definition 24, the direct product of *k* class hierarchies is a poset of composite classes. Querying a composition of multiple class hierarchies is almost like querying a poset of classes. However, the structure of a direct product of class hierarchies is more restricted. It is natural to ask whether the results we obtained in Section 6 apply to the case of multiple class hierachies and the answer is that they do hold. It is easy to see that the preferred abstract answer is not unique. We show that there is no efficient algorithm to obtain a preferred abstract answer. For this purpose, we look at a decision version of a simpler problem.

Multiple Class Hierarchies Abstract Answer

INSTANCE: An abstraction $\mathbf{P_A}$ of a direct product of n class hierarchies, $\mathbf{T_i}$ $(1 \le i \le n)$, a set of composite classes $\mathbf{Q}$ where $\mathbf{Q} \subseteq class(\mathbf{P_A})$ and each class in $\mathbf{Q}$ satisfies the query, and a bound $\mathbf{K}' \in Z^+$ (where Z^+ denotes the positive integers).

QUESTION: Is there an abstraction $\mathbf{A}$ of $\mathbf{P_A}$ such that $extension(\mathbf{A}) = \bigcup \mathbf{Q}$ and the number of classes in $\mathbf{A}$ is no more than $\mathbf{K}'$?

If we could demonstrate the Multiple Class Hierarchies Abstract Answer problem is NP-complete, we would know that its corresponding optimization problem, namely, finding an abstraction with the minimum number of classes for describing $\mathbf{Q}$, is at least as hard.

Theorem 6: Multiple Class Hierarchies Abstract Answer is NP-complete.

Proof: It is easy to see that Multiple Class Hierarchies Abstract Answer $\in$ NP since a non-deterministic algorithm need only guess an abstraction $\mathbf{A}$ and check in polynomial time whether $extension(\mathbf{A}) = \bigcup \mathbf{Q}$ and has the appropriate number of classes.

The remaining part of the proof involves the transformation of some already known NP-complete problem to our problem and showing that the transformation can be done in polynomial time. Below we define the Minimum Cover problem which is known to be NP-complete [KARP72] and then continue with a construction that transform Minimum Cover to Multiple Class Hierarchies Abstract Answer.

Minimum Cover

INSTANCE: Collection C of subsets of a finite set S, positive integer $K \le |C|$.

QUESTION: Does C contain a cover for S of size K or less, i.e., a subset $C' \subseteq C$ with $|C'| \le K$ such that every element of S belongs to at least one member of C'?

Our transformation basically follows that of the proof that the Minimal Disjunctive Normal Form problem is NP-complete [GIMP65]. Let $S = \{s_1, \ldots, s_n\}$ be a finite set, $C = \{c_1, \ldots, c_m\}$ be a collection of subset of S, and a positive integer $K \le m$ be an instance of Minimum Cover. We must construct an abstraction $\mathbf{P_A}$ of a direct product of n class hierarchies $\mathbf{T_i}$ $(1 \le i \le n')$, a set of composite classes $\mathbf{Q}$ where $\mathbf{Q} \subseteq class(\mathbf{P_A})$, and a positive integer $\mathbf{K}'$ such that an abstraction $\mathbf{A}$ of $\mathbf{P_A}$ satisfying $extension(\mathbf{A}) = \bigcup \mathbf{Q}$ contains no more than $\mathbf{K}'$ composite classes if and only if C has a cover of size K or less.

First, we construct n class hierarchies $\mathbf{T_i}$ $(1 \le i \le n')$, with r_i's as the roots, and x_i and x_i' as the only successors for each root. Class hierarchies $\mathbf{T_{n+1}}$ and $\mathbf{T_{n+2}}$ are hierarchies of 3 classes, with p_1, p_2 as their roots, and y_1, y_1' and y_2, y_2', respectively, as their successors. They are named differently because, as we will see later, they serve different purposes. We will be dealing with composite classes over n' (=n+2) class hierarchies. For convenience, we will simply refer to them as composite classes and those which compose of no root classes as *base* classes.

For each element $s_i \in S$, we associate it with a base class

$$\omega_i = \langle x_1, \ldots, x_i', \ldots, x_n, y_1, y_2 \rangle$$

For each $c_i \in C$, we associate it with a composite class

$$\Omega_j = \langle z_{j_1}, \ldots, z_{j_n}, y_1, y_2 \rangle \text{ where } z_{j_k} = \begin{cases} x_k & s_k \notin c_j \\ r_k & s_k \in c_j \end{cases}$$

It should be clear from Definition 24 that if $s_i \in c_j$, $\omega_i \le \Omega_j$; but there are base concepts $\omega \le \Omega_j$ such that $\omega \ne \omega_i$ for all i $(1 \le i \le n)$. Let D be the subset of all base concepts such that $\omega \in D$ iff $\omega \le \Omega_j$ for some j $(1 \le j \le m)$ and $\omega \ne \omega_i$ for all i $(1 \le i \le n)$. Define

$$D_{even} = \{ \langle z_1, \ldots, z_n, y_1, y_2' \rangle \mid \omega = \langle z_1, \ldots, z_n, y_1, y_2 \rangle \in D \text{ and number of non-primed x's is even} \}$$

$$\Omega_{even} = \{ \langle z_1, \ldots, z_n, y_1, p_2 \rangle \mid \langle z_1, \ldots, z_n, y_1, y_2' \rangle \in D_{even} \}$$

$$D_{odd} = \{ \langle z_1, \ldots, z_n, y_1', y_2 \rangle \mid \omega = \langle z_1, \ldots, z_n, y_1, y_2 \rangle \in D \text{ and number of non-primed x's is odd} \}$$

$$\Omega_{odd} = \{ \langle z_1, \ldots, z_n, p_1, y_2 \rangle \mid \langle z_1, \ldots, z_n, y_1', y_2 \rangle \in D_{odd} \}$$

Notice that every base concept in D_{even} (D_{odd}) is contained exactly by one composite class in Ω_{even} (Ω_{odd}). Moreover, every base class in D is also contained by exactly one composite classes in either Ω_{even} or Ω_{odd}. The even-odd arrangement and the involvement of two more class hierarchies, $\mathbf{T}_{n+1}$ and $\mathbf{T}_{n+2}$, are to ensure that each newly introduced composite class has to be included in the abstraction **A** if $D \subseteq \mathbf{Q}$.

Let us complete the construction by specifying **Q** and $\mathbf{K}'$.

$$\mathbf{Q} = \bigcup \{ D \cup D_{even} \cup D_{odd} \cup \{ \omega_i \mid 1 \le i \le n \} \}$$

$$\mathbf{K}' = K + |D|$$

Since our construction basically follows that from [7], we omit showing that the transformation can be done in polynomial time.

We claim that there is an abstraction **A** of $\mathbf{P_A}$ with no more than $\mathbf{K}'$ composite classes if and only if C has a cover of size K or less. Suppose C has a cover size of K or less. We can form an abstraction **A** such that for all composite classes $\Omega \in \Omega_{even} \cup \Omega_{odd}$, Ω is a composite classes in **A**. Notice that $\bigcup \{ \Omega_{even} \cup \Omega_{odd} \} = \bigcup \{ D \cup D_{even} \cup D_{odd} \}$ and $|\Omega_{even}| + |\Omega_{odd}| = |D|$. Now all that remains of **Q** is $\{ \omega_i \mid 1 \le i \le n \}$. Recall that $\omega_i \le \Omega_j$ if $s_i \in c_j$. Since C has a cover of size K or less, we can form an abstraction **A** with no more than $\mathbf{K}' = K + |D|$ composite classes.

Suppose an abstraction **A** has no more than $\mathbf{K}'$ composite classes. By way of construction, for all composite classes $\Omega \in \Omega_{even} \cup \Omega_{odd}$, Ω is a composite classes in **A** and $|\Omega_{even}| + |\Omega_{odd}| = |D|$. Finally, for each Ω_j in **A**, we pick the corresponding c_j for the cover C'. Thus there is a cover C' with size $K = \mathbf{K}' - |D|$ or less. ■

8 Related Work

Many researchers have recognized the need for higher-level, intensional, implicit, query answers. In the context of deductive database, Cholvy and Demolombe [3] choose to ignore the set of elementary facts completely. They were not interested in a set of atomic objects which satisfy a given query in a particular database state, but in the conditions objects must satisfy, in any state, to belong to the answer. The rule base is considered as a set of axioms *T* which are formulas of a first order language. The answer to a query *Q(X)* is a formula *ans(X)* such that

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

that is, $\forall X (ans(X) \rightarrow Q(X))$ is a theorem of *T*, where *X* is a tuple of free variables. An interesting idea is allowing the user to restrict the representation of the answer to a pre-specified set of predicates, which corresponds to the domain of interest to the user. Part of our work here is related to this last suggestion as we assume the availability of pre-defined hierarchies of classes, but our answers to query are expressed not simply as collections of classes but as forests of class hierarchies. The notion of an abstraction, an important concept in this paper, was not addressed by Cholvy and Demolombe.

In a large knowledge base system, data can be represented in the form of both general rules (Horn clauses) and assertions representing specific facts (tuples of relations). Imielinski [8] argues that rules as well as atomic facts should be allowed in the answer for a query. He proposes rule transformation as a technique for query processing. In our case, it is possible to encode a class hierarchies using a set of rules. Since each class is basically a predicate, a class *C* is subsumed by another class *C'* can be expressed as

$$C(x) \rightarrow C'(x)$$

If *Q* is the condition of a query, and the concepts $C_1, C_2, \ldots, C_n$ satisfy the query. The answer can be expressed as a set of rules:

$$\begin{gathered} C_1(x) \rightarrow Q(x) \\ C_2(x) \rightarrow Q(x) \\ \ldots \\ C_n(x) \rightarrow Q(x) \end{gathered}$$

Further, in [4], Chomicki and Imielinski define a formal notion of finite representation of infinite query answers in an extension of Datalog. Again the notion of an abstraction has not been addressed. Moreover, since rules are given as Horn clauses, literals are all positive and thus, exceptions to classes cannot be expressed in the answer.

Taking a TTL catalog as an example, Corella [5] showed the existence of a class of potential applications which are special in that the entities to be retrieved in response to queries are concepts whose extensions are immaterial. A set of “retrievable concepts” is defined from which the elements of all result answers should be chosen. A query has a condition Q, which is a predicate. A concept C satisfies a query with condition Q if Q subsumes C. Concepts are also organized into a finite tree taxonomy. Our definitions of classes and class hierarchies actually correspond to the respective definitions of concepts and taxonomy. There is also a notion of levels of abstraction associated with queries and answers. In Corella’s paper, a level of abstraction is formally defined as a set of concepts; whereas in our definition, an abstraction is a forest of class hierarchies. The motivation is that our definition more readily models a graphical querying and answering facility. We also introduce a labeled abstraction which is unique in this paper. Labeled abstractions increase our expressive power and, further, allow us to express exceptions to an answer.

In a preliminary version [15] of this paper, we were concerned with the implicit representation of extensional answers. Answers are given in terms of expressions of concepts and individuals. Exceptions within individual concepts are allowed. Two criteria are defined as measures of the goodness of such expressions: (i) minimizing the number of terms; (ii) positive terms preferred over negative terms. In this paper, we present a comprehensive study of querying a tree-structured class hierarchy. We introduce the notion an abstraction. The idea of an abstraction not only applies to the presentation of answers, but also applies to the querying for abstract answers. To accommodate exceptions, we further introduce a labeled abstraction. We illustrate that these definitions easily translate into graphical interfaces. Moveover, we consider other sets of criteria that might be preferable, study the uniqueness properties under such criteria and provide efficient algorithms, whenever possible, for obtaining such answers.

Siegelmann and Badrinath [16], based on the preliminary version [15] of this paper, suggest a different set of criteria for measuring the goodness of implicit answers. Their i-coherent expression is an expression that contains no subtree with more than i negative terms, but contains at least one subtree with i negative terms. Each subtree also contains the root term as the only positive term. The user is allowed to choose the parameter i. Coherent answers are not necessarily optimal in the number of terms, but they still are unique in their representation. Our preferred restricted answer expression is similar to i-coherent expression, although the former has its emphasis on the optimal number of terms. Again our comprehensive treatment of the abstraction concept is unique in this paper.

9 Conclusions

Class hierarchy is an important data modeling concept in today's object oriented databases. In this paper, we integrated the class hierarchy concept in the formulation of queries and the presentation of answers. We defined the notion of an abstraction. Informally, the hiding of classes from a class hierarchy forms an abstraction. We applied the concept of an abstraction in a graphical query language. A user can hide some uninteresting classes from the class hierarchy. The representation of the answer is also expressed as an abstraction. The hiding of more classes from the answer implies the suppression of more details and the more abstract the answer is. Thus classes subsumed by other class in the answer are not included. This defined our preferred abstract answer. We proved that preferred abstract answer had the minimum number of classes in its representation and was unique in its representation. We also presented an efficient algorithm for obtaining a preferred abstract answer.

To accommodate exceptions in an abstract answer, we introduced labeled abstraction. Associated with each class, a "+" label indicates that the class is in the answer; whereas, a "-" label indicates the class is not part of the answer. The label for a subclass overrides the label for its superclass. First, we considered a restricted form of abstract answer using labeled abstraction in which every "-" label class has exactly one "+" label ancestor class. The preferred restricted form has the minimum number of classes in its representation and for two representations with the same minimum number of classes, it has fewer "-" labeled classes. We proved that the preferred restricted form was unique in its representation. Again we presented an efficient algorithm for obtaining a preferred restricted form answer. Next, we considered the general abstract answer using labeled abstraction. The criteria for preferred general answer are the same as those for preferred restricted form. We proved that the preferred general answer was still unique in its representation. An efficient algorithm for obtaining preferred general answers can be elicited from the constructive proof.

We extended the idea of an abstraction beyond a tree-structured class hierarchy to a partially ordered set of classes. We illustrated that preferred abstract answer under a poset of classes was no longer unique in its representation. We also proved that obtaining preferred abstract answer under a poset of classes was an NP-complete problem. For multiple class hierarchies, a restricted poset of classes, the problem still remains intractable.

The various set of criteria we considered here as preferred answer are, by no means, exhaustive. Other reasonable criteria include putting a bound on the maximum number of negative terms in the abstract answer, and assigning weights to classes at different levels in a class hierarchy.

Throughout the paper, we assume that for a class to satisfying a query, each object within the class has to be tested individually. An interesting question is to what extent the finding of an abstract answer can be integrated with conventional query optimization and, as a result, be obtained more efficiently.

References

[1] A. Alashgar, S. Su, H. Lam, "OQL: A query language for manipulating object-oriented databases," *Proc. 15th Int. Conf. on VLDB*, Amsterdam, Aug. 1990, pp. 433-442.

[2] J. Banerjee, W. Kim, K. Kim, "*Queries in object-oriented databases,"* Proc. 4th IEEE Conf. on Data Engineering, Feb. 1988, pp. 31-38.

[3] L. Cholvy, R. Demolombe, "Querying a rule base," *Proc. 1st Int. Conf. on Expert Database Systems*, South Carolina, 1986.

[4] J. Chomicki, T. Imielinski, "Finite representation of infinite query answers," *ACM Trans. on Database Systems*, vol. 18, no. 2, Jun. 1993, pp. 181-223.

[5] F. Corella, "Semantic retrieval and levels of abstraction," *Expert Database Systems: Proc. 1st Int. Workshop on Expert Database Systems,* L. Kerschberg (eds.), Benjamin Cummings, New York, 1986, pp. 91-114.

[6] G. Gambosi, M. Scholl, H.-W. Six, eds., *Geographic database management systems: workshop proceedings, Capri, Italy, May 1991*, Springer, 1992.

[7] J. Gimpel, "A method of producing a boolean function having an arbitrarily prescribed prime implicant table," *IEEE Trans. Computers*, vol. 14, no. 3, 1965.

[8] T. Imielinski, "Intelligent query answering in rule based systems," *J. Logic Programming*, vol. 4, no. 3, Sep. 1987.

[9] W. Kim, J. Banerjee, H.-T. Chou, J.F. Garza, "Object-oriented database support for CAD," *Computer Aided Design*, vol. 22, no. 8, Oct. 1990, pp. 469-479.

[10] W. Kim, F. Lochovosky, *Object-Oriented Concepts, Databases, and Applications,* ACM Press, 1990.

[11] L.-C. Liu, E. Horowitz, "Object database support for CASE," *Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD*, R. Gupta, E. Horowitz (eds.), Prentice-Hall, 1991.

[12] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, 1991,

[13] N. Qazi, M. Woo, A. Ghafoor, "A synchronization and communication model for distributed multimedia objects," *Proc. First ACM Int. Conf. on Multimedia*, Aug. 1993, pp. 147-156.

[14] G. Shaw, S. Zdonik, "A query algebra for object-oriented databases*," Proc 6th IEEE Conf. on Data Engineering*, Feb. 1990.

[15] C.-D. Shum, R. Muntz, "Implicit Representation of extensional answers," *Proc 2nd Int. Conf. on Expert Database Systems*, Apr. 1988, pp. 257-273.

[16] H. Siegelmann, B. Badrinath, "Integrating implicit answers with object-oriented queries," Proc 17th Int. Conf. on VLDB, Barcelona, Sep. 1991, pp. 15-24.

[17] S. B. Zdonik, D. Maier, *Readings in Object-Oriented Database Systems,* Morgan Kaufmann, 1990.