# Integrating Implicit Answers with Object-Oriented Queries

**Hava T. Siegelmann**
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
siegelma@paul.rutgers.edu

**B. R. Badrinath***
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
badri@cs.rutgers.edu

## Abstract

Queries in object-oriented databases are formulated against a class and retrieve instances of the class satisfying a certain predicate on the attributes of the class. The presence of a class hierarchy, an integral part of any object-oriented data model, allows answers to be expressed implicitly in terms of classes and instances. This enables answers to be provided at different levels of abstraction. Shum and Muntz [SM88] presented a way of providing implicit expressions based on a taxonomy defined over the database. The algorithm presented in [SM88] is optimal in the length of the answer but the clarity of the answer is often poor. In this paper, the focus is on *coherent* answers: implicit answers that are not necessarily optimal in the number of terms but are easy to comprehend. We show that a unique coherent answer can be obtained efficiently in a top down manner. Since the objective is to provide coherent answers, and user queries are formulated by means of query operators to access the database, the standard query operators are redefined to obtain and manipulate coherent answers. Coherent answers are useful in coping with information complexity as they allow answers to be represented abstractly and are also a useful representation tool for complex information systems.

## 1 Introduction

Object-oriented databases are being developed to support complex data-intensive applications [Kim90, Deu90, Mai89]. Issues in the design of these information systems capable of supporting a rich collection of sophisticated data modeling concepts have become an important area of research [ZM90, KL90, Bee89]. Recent research efforts have concentrated on providing facilities for querying object-oriented data models [Deu90, ASL90, KM90, SZ90, BKK88]. Most of these proposals for query processing in object-oriented databases are based on extending relational framework to support various concepts of object-orientation [Deu90, SZ89, SZ90].

*Supported in part by Henry Rutgers Research Fellowship Award

Typically, an answer to a query is a set of objects satisfying certain predicates [BKK88, SZ89]. This type of answer does not necessarily provide a rich expressive technique nor is an efficient means of presenting complex information [SM88, Cor85]. For example, consider a personnel database of a large corporation and the query:

> Select all employees with a salary greater than 30K

The conventional answer is formed as a set of employee objects whose salaries are greater than 30K. The objects can either be retrieved as object identities or as values of all the attributes of the object instances. If there is a large number of employees whose salaries are more than 30K, and it turns out that all engineers and all managers make more than 30K and also there are two secretaries who earn more than 30K, an elegant answer can be:

> engineers + managers + objid(secretary1) + objid(secretary2).

This assumes that "engineers" and "managers" are predefined classes. Such an answer is called an *implicit expression* or an *implicit answer*. We will use the words expression and answer interchangeably. We call the objects (represented by object ids) that appear in the conventional answer as *instances*. Groups of instances such as engineers and managers are referred to as *classes*. Some of the advantages of an implicit answer are: it does not require explicit enumeration of all the instances. This is useful in meeting resource constraints including those imposed by user interfaces. Further, when complex information is presented, the user may not be interested in too fine a detail. Hence, implicit answers can be used to allow information exchange at higher levels of abstraction; a useful capability in decision support systems. Also, implicit answers help the user in acquiring some global understanding of the answer. Finally, updates at a lower level of abstraction do not conflict with retrievals at a higher level of abstraction. Hence, there exists a potential for enhanced concurrency [MGG86, BR90].

Since we are interested in providing implicit answers to users' queries, it is necessary to design operators capable of providing implicit answers. To this end, we have defined a basic set of query operators based on the relational types of algebraic operations. There are two main contributions in this paper: first, the characterization of implicit answers that are coherent, i.e., easy to comprehend, and a top down algorithm to obtain such answers. Second, the development of a basic query algebra that can efficiently provide coherent answers.

The remainder of this paper is organized as follows. Section 2 deals with related work both in the area of implicit answers in databases as well as query algebra for object oriented databases. In Section 3, we develop the notion of coherent expressions. First, a general definition of implicit answers based on [SM88] is presented. Second, the definition of a coherent expression and the motivation for it are presented. Finally an algorithm to obtain the coherent expression is described. We also prove the uniqueness of the coherent answer. Section 4 presents a query algebra defined to obtain and manipulate coherent expressions in object-oriented databases. Conclusions and future work are discussed in section 5.

## 2 Related Work

The notion of information being retrieved as concepts, as opposed to collection of objects, has been investigated in [Cor85, SM88]. In [SM88], a notion of implicit expression on a database with a taxonomy of concepts was developed. That is, providing an answer to a query, in an implicit, short way rather than a list of objects. The implicit expression uses names of predefined classes of objects and individual objects. Implicit expressions that satisfy certain goodness criteria, called optimal expressions were introduced, that is, the expression with the minimum number of names of classes or objects. Further, a polynomial time algorithm to find the optimal expression after the answer to a query was obtained as a list of objects is also presented. They also provided a proof to show that it is a NP complete problem to obtain an optimal expression during query processing.

In our research, we have used a model that is based on the definitions of implicit expressions given in [SM88]. However, we have defined a different notion of implicit expression called i-coherent expression. The value of i is a "simplicity" parameter of the expression. We allow the user to chose this parameter. There is a trade-off between the simplicity of the expression and its length. The i-coherent expressions can be obtained efficiently during query processing. We have also defined a basic set of query operators capable of providing coherent answers.

Query languages based on extending the relational framework have been proposed for object-oriented databases in[KP90, Deu90, SZ90, SZ89, BKK88]. In [SZ89, SZ90], a query algebra synthesizing relational algebra ideas with object-oriented data concepts was described. In [SZ89], modified operations for nested sets and operators with functions returning objects of a specified type are provided. Though the query algebra provides type specific operation against collections, the answer is still retrieved as a set of objects. The model described in [KP90] is based on supporting nested relations, i.e., the attributes of a tuple can either be atoms or relations. A nested relational algebra is proposed to provide greater expressive power to deal with hierarchical structure of data.

We have developed query operators to manipulate coherent expressions in an object-oriented framework. However, we have assumed a simple model where the attributes are primitive objects.

## 3 Coherent Answers

The object-oriented data model uniformly models real world entities as objects. In addition, similar objects are grouped into classes that describe the behavior of objects. Classes are organized in an inheritance hierarchy. Some models provide multiple inheritance and the domain of the attributes can be classes; thus, supporting complex objects. However, in this paper, we restrict our discussion to a taxonomy and the attributes to be primitive objects. The concepts and ideas in this paper can be extended to a more general object-oriented data model that includes complex objects and multiple inheritance. This will be reported elsewhere.

The objective of our research is to make use of the inheritance hierarchy and provide implicit expressions (answers in terms of classes and instances) as opposed to answers at a single level of abstraction, i.e., an exhaustive list of instances. We consider a finite set $D$ of objects: $D = \{D_1, D_2, \ldots\}$. $C = \{C_1, C_2, \ldots\}$ is the set of classes relative to $D$. An *entity* is either an instance or a class. We do not deal with an arbitrary collection of instances but with a tree-structure of classes, namely a taxonomy. A *taxonomy* is a finite tree whose nodes are instances and classes. Instances are leaves and internal nodes including the root are classes. The successor of each node is subsumed by its parent class. The union of all successors of any non-leaf node is equal to the parent class. All siblings are mutually exclusive. A set of instances A, is *classifiable* by a taxonomy T iff the root of T contains A.

An example illustrating a taxonomy is shown in Figure 1. Figure 1 is an example of a vehicle class. This example will be used to illustrate concepts and ideas presented in this paper. Classes are shown in ellipses and instances are shown in rectangles. In Figure 1, land-vehicle, water-vehicle, etc. are classes. The set of all instances of vehicles (shown in rectangles) is classifiable by the vehicle schema. The common subset of attributes for all the instances are year, color, owner,

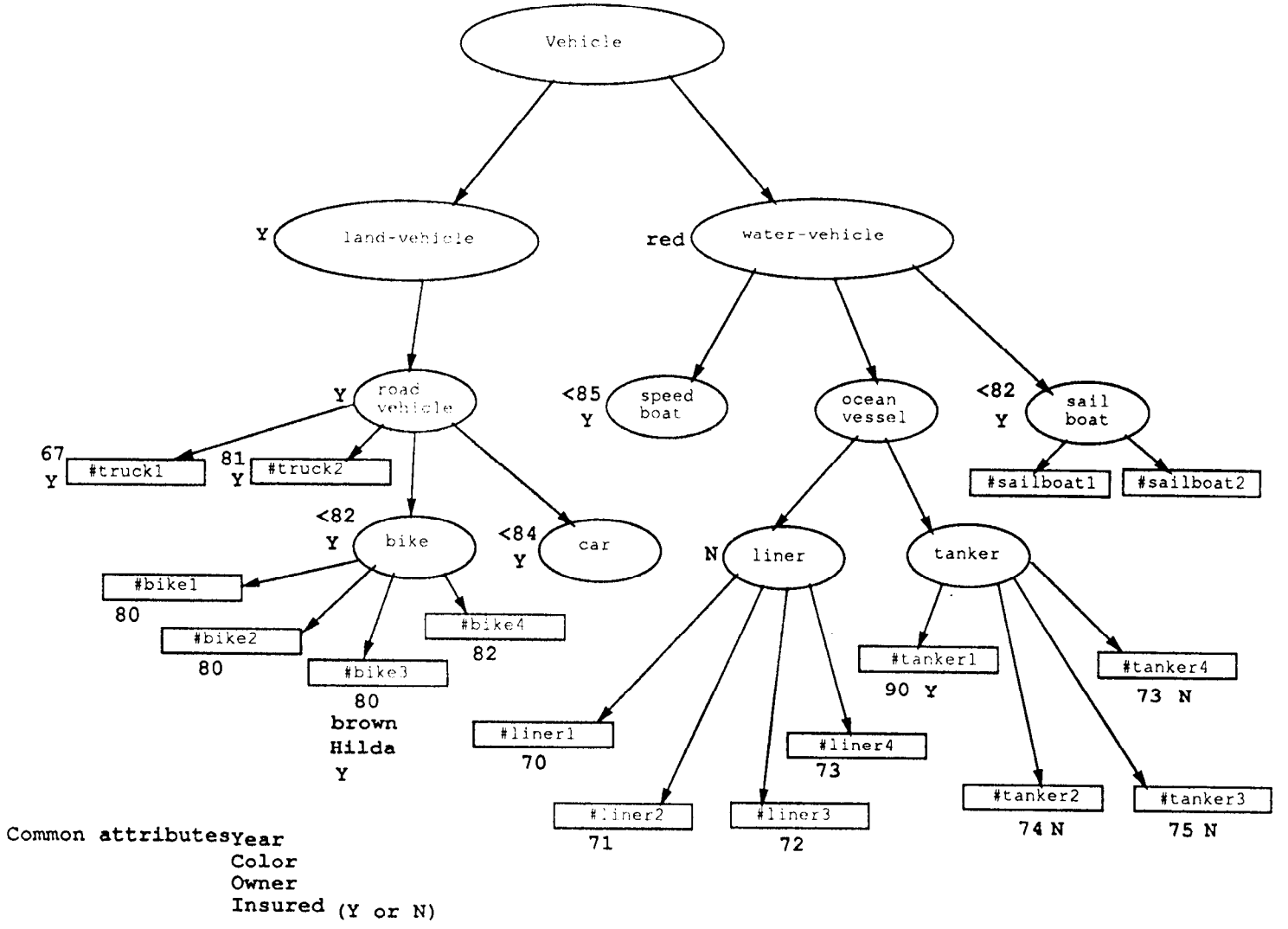



Figure 1: Vehicle Schema with attribute values

and insured. The domain of these attributes are either strings or integers. The values of the attributes are shown by the side of the instances, and constraints on the values are shown by the side of the classes.

### 3.1 An implicit expression

A conventional answer to a query consists of a set of instances satisfying a predicate. However, we need to find a way of expressing answers in terms of classes and instances. Hence, we need to explore the notion of implicit expressions. With respect to the vehicle schema of Figure 1, the following are answers that imply instances but contain both classes and instances; instances are represented by their object ids.

**Example 1** Find all water-vehicles that were manufactured before Year 1989. For this query, examples of implicit expressions are:

1. water-vehicle - #tanker1
2. ocean-vessel + sail-boat + speed-boat - #tanker1
3. (water-vehicle - ocean-vessel) + liner + (tanker - #tanker1)

Thus, an implicit expression consists of classes and instances. However, the set of instances implied by these expressions is often difficult to deduce and to comprehend. Hence, we need a characterization of implicit expressions that are clear to the user. In order to do this, we need to define some terms and concepts.

**Definition 1:** The *alphabet* of an expression defined over a taxonomy T is composed of the following:

- Classes: $C_1, C_2, \ldots$
  Each class is a label of a node in T.
- Instances: $D_1, D_2, \ldots$
  Each instance is an element of the root class of T.
- Empty: $\epsilon$
  This denotes the empty expression.
- Signs: $+, -$.
  The sign + denotes inclusion, while the sign − denotes set difference
- Parenthesis: (,).
  They can be added for clarity.

**Definition 2:** An *expression* over the taxonomy T is defined inductively as follows:

- An entity (either an instance or a class) is an expression.
- $\epsilon$ is an expression.
- If e is an expression so is (e).
- If $e_1$ and $e_2$ are expressions, so are $e_1 + e_2$, $e_1 - e_2$.

**Definition 3:** The *meaning* of an expression over a taxonomy T is the set of instances (zero or more) that are represented by the expression. The meaning of an expression is evaluated from left to right and is not associative. We assign it as meaning( exp ). Two expressions $e_1$, $e_2$ are *equivalent* if meaning($e_1$) = meaning($e_2$).

**Definition 4:** A *path* in the taxonomy is a list of successive nodes starting with the root and ending with a leaf. A *subexpression* of an expression e over a path P in the taxonomy is an expression formed by removing from e all those entities that are not in P

**Definition 5:** The *length* of an expression is the number of entities that appear in it. The *size* of an expression is the number of instances it implies.

### 3.2 Towards a coherent expression: Definitions and examples

In this section, we will define certain constraints on implicit expression so that they are easy to comprehend. Such expressions are called coherent expressions. We will construct the coherent expression in two stages. First, a notion of simple expression is introduced and then coherent expression is defined by specifying the desired characteristics. Both expressions are implicit expressions with certain constraints. They both consist of two types of terms:

**Definition 6:** A *term* is a sequence of entities with one of two possibilities:

a. $(C_i - D_{i1} - \ldots - D_{il})$ $D_{ij} \in C_i$ and $l \geq 0$. This term is named *a conceptual term* $C_{term}$

b. $D_k$, this is the *individual term* $I_{term}$, or an *instance*

**Definition 7:** A *simple expression* is an implicit expression consisting of the union of conceptual terms and individual terms with the following conditions:

a. No entity appears more than once in the expression

b. For every path $P$, the subexpression over $P$ includes at most one class

c. For any $I_{term}$ and $C_{term}$, $I_{term} \in C_{term}$, if $I_{term} \in expression \Rightarrow C_{term} \notin expression$.

d. Entities appear in left to right order of the taxonomy (for uniqueness).

**Example 2** Consider the query to find all ocean-vessels that were manufactured between the years 1971 and 1975. Equivalent simple expressions for this query are:

1. #liner2 + #liner3 + #liner4 + #tanker2 + #tanker3 + #tanker4
2. (ocean-vessel - #liner1 - #tanker1)
3. (liner - #liner1) + (tanker - #tanker1)

**Definition 8:** A *coherent expression* is a simple expression with the following constraints:

a. does not contain classes $C_1, C_2, \ldots, C_k$, $k > 1$, s.t. $meaning(C_1 + C_2 + \ldots + C_k) = meaning(C_j)$ for some $C_j$ class belongs to T

b. no conceptual term contains $C_i$ such that there exists $C_j$ with equivalent meaning, and $C_j$ is an ancestor of $C_i$ in T

c. does not contain $C_{term}$ such that there is an equivalent sequence of instances that is not longer than it

d. does not contain a sequence of instances when there is an equivalent $C_{term}$ that is shorter than the sequence

**Example 3** Example of coherent expressions

1.(ocean-vessel - #liner3) + (speedboat) +(sailboat) is not a coherent expression because of (a)
2.(water-vehicle - #liner3) is a coherent expression
3.(road-vehicle - #bike2) is not a coherent expression because of (b)
4.(land-vehicle - #bike2) is a coherent expression
5.(tanker - #tanker3 - #tanker4) is not a coherent expression because of (c)
6.#tanker1 + #tanker2 is a coherent expression
7.#tanker1 + #tanker2 + #tanker3 is not a coherent expression because of (d)
8.(tanker - #tanker4) is a coherent expression

Conceptual terms with a large number of negative instances tends to make an implicit expression incoherent. However, there is a trade-off between the length of the conceptual terms and the total length of the expression. We allow the user to decide the maximum number of negative instances in $C_{term}$.

**Definition 9:** An *i-coherent expression* is a coherent expression that does not contain a $C_{term}$ with more than i negative instances, but contains at least one $C_{term}$ with i negative instances.

Notice that the definition of the i-coherent expression makes the characteristics (a) and (d) of a coherent expression valid only if the corresponding $C_{term}$ does not contain more than $i$ negative instances.

**Example 4** Consider the query to find all ocean-vessels that were manufactured between the years 1972 and 1975. Examples of equivalent i-coherent expressions for this query are:

1. (ocean-vessel - #liner1 - #liner2 - #tanker1)
   3-coherent expression
2. #liner3 + #liner4 + (tanker - #tanker1)
   1-coherent expression

### 3.3 Features of the i-coherent expression

In this section, we will prove that i-coherent expressions are unique, i.e., if $e_1$ and $e_2$ are i-coherent and meaning($e_1$) and meaning($e_2$) are equivalent, then $e_1 = e_2$. Further, given a taxonomy, it is shown that i-coherent expressions can be obtained top down in polynomial time. Notice that there can be an i-coherent expression and a j-coherent expression for $i \neq j$ that are equivalent.

**Theorem 1:** An i-coherent expression is unique.

*Proof:*
To aid in the proof, we define a new term called $I_{max}$ term. This term is formed by grouping the maximum number of adjacent $I_{term}$s in the i-coherent expression such that $I_{max}$ term has a class in the taxonomy that includes all the grouped elements but includes no other entity from the i-coherent expression.

Assume two i-coherent expressions, $e_1$ and $e_2$, that differ at least in one term. Without loss of generality, assume that these expressions differ in their left most term. We will call them $term_1$ and $term_2$.

Notice that two terms derived from the same target class can either subsume the meaning of the other, or be equivalent in meaning. They can not be overlapping because of the tree structure of the taxonomy.

1. meaning($term_1$) subsumes meaning($term_2$):
   - if they are both conceptual terms, then $term_2$ is not the highest class that is possible, and thus violates (b)
   - if $term_1$ is a $C_{term}$ and $term_2$ is an $I_{max}$ term, then meaning of $term_1$ subsumes the meaning of at least one more conceptual term of $e_2$ and thus $e_2$ violates (b)
   - if $term_1$ is an $I_{max}$ term, part of it is equivalent to a conceptual term in $e_2$; this can not be the case (see next part of the proof).

2. meaning($term_1$) and meaning($term_2$) are equivalent in their meaning:
   - if both are $C_{term}$s then it is the case where a class in the taxonomy has only one child. The term with a lower class violates (b).
   - If $term_1$ is an $I_{max}$ term and $term_2$ is a $C_{term}$, If $term_2$ is not shorter than $term_1$, $e_2$ violates (c) else $e_1$ violates (d).
   - if both are $I_{max}$ terms, they must be identical, else their meanings are not equivalent. □

For a given query, an i-coherent expression is obtained in two phases. In the first phase, for each class, the number of positive instances and the number of negative instances for the query is found. This is done by a depth first search on the taxonomy starting from the root and

determining which instance satisfies the query. The second phase is described in the following algorithm. At each node of the tree, the algorithm either returns a term (conceptual or individual) or proceeds recursively. The final answer is the concatenation of these terms and is the maximum possible j-coherent expression for $j \leq i$.

0. current-node $\leftarrow$ the root.
1. if current-node is a leaf (an instance)
   return instance if it satisfies the query,
   else return $\epsilon$.
2. if current-node has only one child
   current-node's children $\leftarrow$ grandchildren
3. if all the instances under current-node do not satisfy the query
   return $\epsilon$.
4. assume p instances under current-node satisfy the query and n instances under current-node do not satisfy the query.
   if $n \leq i$ (i-conceptual term or less) and
   $p > (n+1)$ (The conceptual term is shorter than the equivalent sequence of instances)
   return (current-node $-D_{j1} - D_{j2} \ldots - D_{jn}$)
5. If $n > i$ or $p \leq (n+1)$
   apply the algorithm to all the children of the current-node.

Now, we will give a proof of correctness for the above algorithm.

**Theorem 2:** An i-coherent expression can be obtained in two phases. The first phase in time O(d + c), d is the number of instances and c is the number of classes in the schema and the second phase in time O(hA), where h is the height of the taxonomy and A is the size of the answer set.

*Proof of Theorem 2:*

The proof lies in correctness of the algorithm and its complexity.

**Correctness:** Note that each term in the i-coherent expression is formed either in step 1 or step 4.

**a.** The expression obtained by the algorithm is simple: By step 4, while returning a term at node $C_j$ the algorithm will not proceed to $C_j$'s children.

**b.** No $C_{term}$ has more than i negative instances. A conceptual term cannot have more than i negative instances due to step 4.

**c.** Condition (a) of coherent expression. The proof is contained in the following lemma.

**Lemma 1**

If the processing continues at a current-node's sons because $p \leq (n+1)$ (while $n \leq i$) than in the final answer there is no set of classes $C_1, C_2, \ldots, C_l$, $l > 1$, that meaning$(C_1 + C_2 + \ldots + C_l)$ is equal to meaning(current-node).

*Proof:* Assume current-node has t sons and $t > 1$ (because if there is only one child, then step 2 will be executed).


1. If some of $S_k$'s are instances then there does not exist a set of classes in the expression such that the sum of their meaning is equivalent to that of the current-node.
2. If $S_k$, $1 \leq k \leq t$, are classes, the following lemma gives a proof:

   **Lemma 2**

   If the processing continues at the current-node's children while $p \leq (n+1)$, $n \leq i$, and $S_k$ are all classes then not all of them will appear in the final answer.

   *Proof:* Assume that all the $S_k$s appear in the final answer. This implies that:

   $p1 > n1 + 1$
   $p2 > n2 + 1$
   .
   .
   .

   which gives $p > n + t > n + 1$ (as $t > 1$), contradicting the assumption $p \leq (n+1)$. □

Lemma 2 implies that there are $S_k$'s that do not appear in the final answer. In these nodes p $\leq$ n +1. Applying Lemma 2 to these nodes ($S_k$'s) implies that instances should be reached, else the tree has an infinite number of levels, contradictory to the definition of a taxonomy as a finite tree. □

**d.** Condition (b) of coherent expression. The conceptual terms include the highest possible classes as the algorithm traverses the classes top-down.

**e.** Condition (c) of coherent expression. In step 5, the algorithm is applied to the children if $n > i$ or p$\leq$(n+1); hence instances are preferred over conceptual terms that are not shorter than the equivalent sequence of instances.

**f.** Condition (d) of coherent expression. Assume that the the final expression contains a sequence of instances $D_1 + D_2 + \ldots + D_k$ and $\exists j$ ($j \leq i$ and $j < k$) such that meaning$(C_m - D_{m1} - \ldots - D_{ij})$ = meaning$(D_1 + \ldots + D_k)$. Since the algorithm executes top down, it has processed the node $C_m$ and has checked the possibility of the term $(C_m - D_{m1} - \ldots - D_{ij})$. The only reason for not

choosing a conceptual term with $C_m$ is ($j > i$ or $j \geq k$). Thus contradicting the assumption.

**Complexity:** First phase of the algorithm takes O(d +c). In the second phase, the algorithm scans only the entities in the paths of those that appear in the answer; hence, the complexity of the second phase is O(h*A) where h is T's height and A is the number of instances implied by the answer.

## 4 Query Algebra

Query evaluation in object-oriented databases requires operators to efficiently access complex structures of data. Query algebras proposed in [KP90, Deu90, SZ89] provide special operators to handle nested sets and other concepts of object orientation. Since coherent expressions can be obtained from a hierarchy in a top down manner, it is possible to integrate the process of finding coherent expressions with conventional query processing.

The algorithm described in Section 3.3 was applied to the root class. However, it can be applied to any other class as well. The set of instances implied by a class, i.e., its leaves, is called a target set. A target set can be given as either a name of a class or implicitly as a coherent expression. Notice that a class name is just a special case of a coherent expression. Thus, we define our query operators on coherent expressions rather than on a set of instances. We will define a basic set of operations: *select*, two types of *project*, and two types of *join*.

### 4.1 The select operator

The *select* operation is defined on either a target-set or a coherent expression. The *select* returns a set of instances that satisfy the selection predicate P. In our case, the select would provide an i-coherent answer.

1. *select* that is defined on a target-set, $select(S, P)$, is calculated by the i-coherent algorithm.
2. *select* that is defined on a coherent expression, e, $select(e, P)$ is calculated as follows:
   $e = term_1 + term_2 + \cdots + term_l$,
   $term_i \in \{I_{term}, C_{term}\}$
   $select(e, P) = \bigcup_{i=1}^{l} select(term_i), term_i \in e$
   - If $term_i$ is $I_{term}$:
     the answer is either the instance itself, if it satisfies P, else $\epsilon$.
   - If $term_i$ is $C_{term} = (C - D_1 - \ldots - D_k)$
     the answer is derived by regarding the instances of C as members of a smaller target set, S(C), and assuming that the "negative instances" of $C_{term}$ are negative examples to the predicate P.

**Example 5**

1. $select(vehicle, year < 1987) = land\text{-}vehicle + (water\text{-}vehicle - \#tanker1)$
2. $select(land\text{-}vehicle + (water\text{-}vehicle - \#tanker1), insured = Y) = land\text{-}vehicle + speedboat + sailboat$

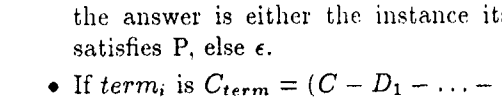
### 4.2 The project operator

The *project* operation returns values of a subset of the attributes for each instance being queried. This subset is specified in the *project* operation. The *project* operation defined on an i-coherent expression, has the form: $project(e, A_1, A_2, \ldots, A_k)$. The attributes, $A_1, A_2, \ldots, A_k$, are a subset of the attributes of the target-set S from which e is derived. We define two types of *project*. In *s-project*, the answer is a set of tuples, each tuple matches the queried attributes of one of the instances of e. Thus *s-project* is the standard project. In *t-project*, the answer is a tuple, where each tuple element consists of all possible values of the attribute in the instances implied by e. This definition of *t-project* can exploit the presence of a taxonomy if constraints on the values of the attributes are specified in the classes.

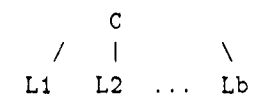
#### 4.2.1 t-project

In the project operator,
$t\text{-}project\ (e, < A_1, A_2, \ldots, A_k >) =< a_1, a_2, \ldots, a_k >$,
each $a_i$ is a set of all the possible values of $A_i$ in the expression e.
*t-project* is evaluated as follows:
$e = term_1 + term_2 + \ldots + term_l$
$t\text{-}project(e, < A_1, A_2, \ldots, A_k >) =$
$\biguplus_{i=1}^{l}(t\text{-}project(term_i, < A_1, A_2, \ldots, A_k >)$
where $\biguplus_{i=1}^{3}(< a1, a2, a3 >, < b1, b2, b3 >) =$
$< \{a1, b1\}, \{a2, b2\}, \{a3, b3\} >$

1. If $term_i = I_{term}$, (an instance D):
   $t\text{-}project(e, < A_1, A_2, \ldots, A_k >) =$
   $< D_{A1}, D_{A2}, \ldots, D_{Ak} >$,
   where $D_{Ai}$ is the value of the attribute $A_i$ of the instance D.
2. If $term_i$ is $C_{term}, (C - D_1 - \ldots - D_m)$: we can not substitute it with either $t\text{-}project(C)$ or $t\text{-}project(C) - t\text{-}project(D_1 + D_2 + \ldots + D_m)$. Consider the following example:




$t\text{-}project((C - D_1 - D_2), color) =< \{red, green\} >$
$t\text{-}project(C, color) =< \{yellow, red, green\} >$
$t\text{-}project(C, color) - t\text{-}project(D_1 + D_2, color) =$
$< green >$
Hence, we have to process *t-project* recursively:




In the above figure, each $Li$ could be either a class or an instance.

$t\text{-}project((C - D_1 \ldots - D_m), < A_1, A_2, \ldots, A_k >) =$
$\biguplus_{i=1}^{b}(t\text{-}project(E_i, < A_1, A_2, \ldots, A_k >)$
where $E_i$ is the expression $L_i \setminus \{D_1 + D_2 + \ldots + D_m\}$, and $L_i$ is a child of C.

3. If $term_i$ is a class C:
The attributes $A_1, \ldots, A_k$ are subset of the attributes of the instances of the target-set S. However, the class C may have constraints on the values for only a subset of these attributes. We examine each attribute separately and insert this value in a tuple.
$t\text{-}project(C, < A_1, A_2, \ldots, A_k >) =$
$\diamond_{i=1}^{k}\ t\text{-}project(C, A_i)$ (forms a tuple out of the $k$ values)
$t\text{-}project(C, A_1) = C.A_1$
if $A_1$ has a constraint at $C$
$\bigcup_{i=1}^{b}\ t\text{-}project(L_i, A_1)$ otherwise

**Example 6**
Let $e =$ ( water-vehicle $-\#tanker2$), then
t-project $(e, < color, insured >) =$
$\diamond(red, \bigcup[t\text{-}project\ ((ocean\text{-}vessel - \#tanker2),$
$< \text{insured} >),$
$t\text{-}project\ (speedboat, < \text{insured} >)$
$t\text{-}project\ (sailboat, < \text{insured} >)]) =$
$\diamond(red, \bigcup[t\text{-}project(liner, < insured >)$
$t\text{-}project((tanker - \#tanker2),$
$< insured >)$
$Y$
$Y]) =< red, \{Y, N\} >$

### 4.2.2 s-project

The definition of this operation is the same as the standard definition of *project*.

$s\text{-}project(e, < A_1, A_2, \ldots, A_k >) =$
$\{< a_1, a_2, \ldots, a_k >\}$
Each tuple consists of the values of the attributes $A_1, A_2, \ldots, A_k$ for the instances of $e$.

## 4.3 The join operator

The *join* operator is used to create relationships between two groups A and B of instances; these instances are implied by i-coherent expressions. The join operation returns a list of new instances that is the cartesian product of A and B. We define two types of *join*: Join between two target sets A and B that have different attributes at the leaf level, called *d-join*. The other join is the special case where all the instances have attributes defined from the same attribute set, called *s-join*.

The *join* operation of two coherent expressions is operated by adding the results of the *join* of the different terms, that it,
$e1 = term_{1,1} + term_{1,2} + \ldots + term_{1,n}$
$e2 = term_{2,1} + term_{2,2} + \ldots + term_{2,m}$,
$terms \in \{C_{term}, I_{term}\}$
$join(e1, e2) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} join\ (term_{1,i}, term_{2,j})$

The concept of *join* on different terms depends upon the type of join.

### 4.3.1 s-join

The *s-join* is an intersection between two groups of instances, each implied by a coherent expression.

1. If one of the terms is an instance:
$s\text{-}join(X, D_2) = D_2, \quad if (X = D_2) or (D_2 \in X)$
$\epsilon, \quad otherwise$

2. If both terms are classes:
$s\text{-}join(C_1, C_2) = C_1, \quad if C_1 \subset C_2$
$C_2, \quad if C_2 \subset C_1$
$\epsilon, \quad otherwise$

3. If both are conceptual terms (and not both classes):
$s\text{-}join((C_1 - D_{1,1} - D_{1,2} \ldots - D_{1,k}), (C_2 - D_{2,1} - D_{2,2} \ldots - D_{2,h})) =$
$s\text{-}join(C_1, C_2) - [D_{1,1} + D_{1,2} \ldots + D_{1,k} + D_{2,1} + D_{2,2} \ldots + D_{2,h} \parallel s\text{-}join(C_1, C_2)]$
where '||' represents the "restricted to" operation, i.e., only the set of instances implied by $s\text{-}join(C_1, C_2)$. At this point the resulting expression may be *2i-coherent*. However, as the expression was obtained from two coherent expressions, it can be modified to an i-coherent by using the algorithm for obtaining i-coherent expression on each term that has more than i negative instances.

**Example 7**
$e1 = car + (tanker - \#tanker1)$
$e2 = (water\text{-}vehicle - \#tanker4)$
$s\text{-}join(e1, e2) = s\text{-}join(car, water\text{-}vehicle) -$
$(\#tanker4 \parallel s\text{-}join(car, water\text{-}vehicle)) +$
$s\text{-}join(tanker, water\text{-}vehicle) - (\#tanker1 + \#tanker4 \parallel s\text{-}join\ (tanker, water\text{-}vehicle)) =$
$\#tanker2 + \#tanker3$

### 4.3.2 d-join

*d-join* is conceptually similar to a *natural join*. The groups of instances A and B implied by the coherent expressions are derived from target-sets having different attributes. Figure 2 is a taxonomy with the animal class as the root. $D_1$ is an instance that belongs to class animal and $D_2$, $D_3$ are instances of the vehicle target class.

1. If both terms are instances:
*d-join* is the same as a regular join.
for example:
$D_1 =$<Tail: 8in, Color: white, Friendly: N >

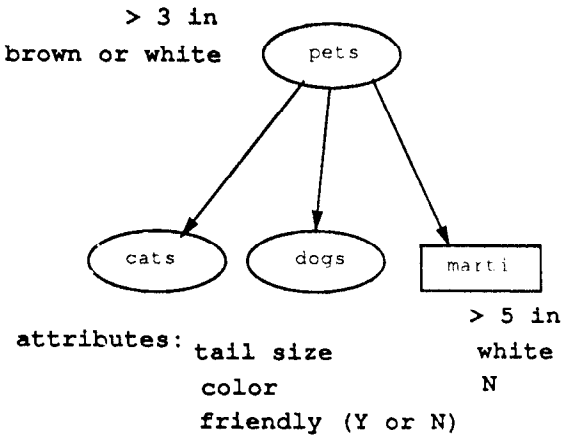



Figure 2: Animal Schema

$D_2$ =<Year: 1988, Insured: Y, Color: white, Owner: john>
$D_3$ =<Year: 1990, Insured: Y, Color: yellow, Owner: john>

*d-join*$(D_1, D_2)$ =
<Tail: 8in, Color: white, Friendly: N, Year:1988, Insured: Y, Owner: john >
*d-join*$(D_1, D_3) = \epsilon$

2. If one of the terms is an instance:
   we can decompose it as:
   *d-join*$((C_1 - D_{1,1} - D_{1,2} \ldots - D_{1,k}), D_2)$ =
   *d-join*$(C_1, D_2)$ − [*d-join*$(D_{1,1}, D_2)$ + ... + *d-join*$(D_{1,2}, D_2) \ldots$ + *d-join*$(D_{1,k}, D_2)$]
3. If both terms are classes with default values: treat the terms as instances.
4. If both terms are conceptual terms:
   *d-join*$((C_1 - D_{1,1} - D_{1,2} \ldots - D_{1,k}), (C_2 - D_{2,1} - D_{2,2} \ldots - D_{2,h}))$ =
   *d-join*$(C_1, C_2)$−
   *d-join*$(C_1, D_{2,1}) \ldots$ + *d-join*$(C_1, D_{2,h}))$−
   *d-join*$(C_2, D_{1,1}) \ldots$ + *d-join*$(C_2, D_{1,k}))$

Note that the expression obtained from a *d-join* is not i-coherent.

**Example 8**

$e1 = (road\text{-}vehicle - \#bike3)$ target-set is vehicle
$e2 = (pets - \#marti)$ target-set is animal
*d-join*$(e1, e2)$ = *d-join*$(road\text{-}vehicle, pets)$−
*d-join*$(\#bike3, pets)$−
*d-join*$(road\text{-}vehicle, \#marti)$ =
$< insured : Y, tail : 3in, color : \{brown, white\} >$
$- < insured : Y, tail : 3in, color : brown, Year : 80, owner : Hilda >$
$- < insured : Y, tail : 5in, color : white, friendly : N >$

## 5 Conclusions and Future Work

In this paper, we explored the problem of providing implicit answers to queries in object-oriented databases. A new definition of implicit expression called a coherent expression was introduced. Coherent expressions are abstract responses that are easy to comprehend. A top down algorithm to obtain such coherent expressions was also described. Since coherent expressions are obtained top down, it was possible to integrate this process with query processing capabilities. To this end, we provided a basic set of operators that returns abstract responses. The capability of obtaining abstract responses from object-oriented databases is a useful tool not only for reasoning about complex information but also for presenting complex information in multi-media databases.

We considered a simple object-oriented model. The concepts and ideas explored in this paper can be applied to a more general object-oriented data model. One such extension is to apply the concept of i-coherent expressions to complex objects, i.e., where the attributes are not atomic but may be classes or target sets. We are currently exploring the issues of providing abstract responses in such object-oriented databases. Another related open problem is to find the characteristics of expressions that can be obtained as part of the query processing in polynomial time (e.g. the coherent expression). We must also investigate other ways of expressing abstract responses; possibly a family of expressions that have some special structure or features that make them amenable to better comprehension.

**Acknowledgments** We would like to thank the referees for their comments and suggestions.

### References

[ASL90] Alashgar, A. M., Su, S. Y. W., and Lam H. OQL: A query language for manipulating object-oriented databases. In *Proceedings of the 15th international conference on VLDB*, pages 433–442, Amsterdam, August 1990.

[Bee89] Beeri, C. Formal models for Object Oriented databases. In *First International Conference on Deductive and object-oriented databases*, pages 370–395, December 1989.

[BKK88] Banerjee, J., Kim, W., and Kim, K. Queries in Object-oriented databases. In *Fourth IEEE Conference on Data Engineering*, pages 31–38. February 1988.

[BR90] Badrinath, B. R. and Ramamritham, K. Performance evaluation of semantics-based multilevel concurrency control protocols. In *Proceedings of the ACM SIGMOD international conference on management of data*, pages 163–172, May 1990.

[Cor85] Corella, F. Semantic retrieval and levels of abstraction. In L. Kerschberg, editor, *Expert Database Systems*. Benjamin Cummings, 1985.

[Deu90] Deux, O. et al. The story of O2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, March 1990.

[Kim90] Kim, W. Object-Oriented databases: Definition and research directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, September 1990.

[KL90] Kim, W. and Lochovosky, F. H. *Object-oriented concepts, databases, and applications.* ACM Press, 1990.

[KM90] Kemper, A. and Moerkotte, G. Advanced query processing in object bases using access support relations. In *Proceedings of the 15th international conference on VLDB*, pages 290–301, Amsterdam, August 1990.

[KP90] Korth, H. and Peltier, X. Query algebra for Object-Oriented databases. In *Proceedings of the Schlumberger software conference*, 1990.

[Mai89] Maier, D. et al. The gemstone data management system. In Won Kim and Frederick H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 283–308. ACM Press, 1989.

[MGG86] Moss, J. E. B., Griffeth, N., and Graham, M. Abstraction in recovery management. In *Proceedings of the ACM SIGMOD international conference on management of data*, pages 72–83, May 1986.

[SM88] Shum, C. and Muntz, R. Implicit representation for extensional answers. In *Proceedings of the Second International Conference on Expert Database Systems*, pages 257–273, Washington, D.C, 1988.

[SZ89] Shaw, G. M. and Zdonik, S. Object-Oriented queries: Equivalence and optimization. In *First International Conference on Deductive and object-oriented databases*, pages 264–278, December 1989.

[SZ90] Shaw, G. M. and Zdonik, S. A Query algebra for object-oriented databases. In *Sixth IEEE Conference on Data Engineering*, February 1990.

[ZM90] Zdonik, S. B. and Maier, D. *Readings in Object-Oriented database systems.* Morgan Kaufmann, 1990.