Using Subsumption in Semantic Query Optimization*

Domenico Beneventano°, Sonia Bergamaschi◇
Stefano Lodi°, Claudio Sartori°

CIOC-CNR
e_mail: {domenico, sonia, lodi, claudio}@deis64.cineca.it

Abstract

Semantic Query Optimization optimize query processing by transforming a query into a semantically equivalent one, i.e. a query whose result is the same for every instance of the database. Semantic query optimization can be obtained by a query rewriting that takes into account the integrity constraints the database must satisfy. In this paper we analyze the possibility of implementing semantic optimization in OODBMs by means of the subsumption relation computation over a database schema. *Subsumption* relations over a schema includes all the specialization relations between classes (types) both explicitly declared in the schema and entailed by the schema. Subsumption computation has been previously used in the database area to perform schemata acquisition and checking for coherence and query processing optimization. In this paper we extend complex object data models with integrity constraints expressed as rules and propose to apply subsumption computation to perform semantic query optimization in object-oriented databases. In this way it is possible to obtain queries having as arguments more specific classes and minimize data access. The paper includes a formalization of complex object data models including path description and integrity constraint rules, enriched with the notion of *subsumption*. A method for semantic query optimization is proposed and its feasibility is analyze by examining some example.

1 Introduction

Semantic optimization is the process which determinate the semantic transformations providing queries with a lower execution cost. Two queries are *semantically equivalent* if their answer is the same for all database state which satisfy a given set of integrity constraints; a *semantic transformation* transforms a query into a semantically equivalent one. Semantic query optimization has been recently faced in Database environment [13, 19, 20] by exploiting first order predicates expressing integrity constraints for query rewriting. The usual environment is that of the relational model [13, 20] where integrity constraints are considered as value restrictions and simple rules on attributes.

In this paper we propose to extend the environment of semantic query optimization to object oriented databases by introducing a general formalism for complex object data models

*This research has been partially funded by the project *Sistemi informatici e Calcolo Parallelo*, sottoprogetto 5, obiettivo LOGIDATA$^+$ of the National Research Council (CNR).

° Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Viale Risorgimento 2,I-40136 Bologna, Italy.

◇ Facoltà di Ingegneria, Università di Modena, Via G. Campi 213/A, I-41100 Modena, Italy.

expressing integrity constraints as simple rules. Furthermore, we propose to exploit *subsumption* relation computation for performing semantic query optimization.

Let us briefly comment the proposed formalism and introduce the subsumption relation computation. The formalism incorporates well known notions, such as those of type, complex values, classes, objects with identity and inheritance, in full analogy with complex objects models recently proposed in the deductive database area [4, 1, 2] and object-oriented database area [3, 16, 15]. The main extensions, as proposed in [10], derived by *terminological logic* languages are: the *conjunction* operator, that allows to express multiple inheritance as a part of a class description; the distinction between *base classes* and *derived classes* (that correspond to primitive and defined concepts in terminological logic languages [12]), where the descriptions of derived classes are interpreted as a set of necessary and sufficient conditions. Moreover, given an extensional semantics to class descriptions, it is possible to evaluate the *subsumption* relations entailed by class descriptions.

Traditionally, in database models, only *base classes* are considered: all class descriptions are interpreted as necessary conditions. A more active role can be played if derived classes are introduced. A new class can be automatically *classified* (namely its right location in the existing taxonomy can be determined) finding the set of its most specific generalizations and of its most generalized specializations. As a result, minimality of a schema with respect to inheritance can be easily computed and inconsistency can be avoided by verifying whether a class is classified below the empty type.

Moreover, classification is important for *query optimization*, as recently discussed in [11, 7, 18]; the proposed idea is that a query can be expressed as a class description which can be classified in the class taxonomy, giving direct access to the set of objects satisfying it.

This paper extends the applicability of such an approach to the query optimization problem in complex objects data models, by exploiting and extending techniques proposed in database semantic query optimization works.

The original contribution of the paper, with respect to the above mentioned query optimization techniques, is the addition to the class descriptions of a set of integrity constraints (domain restrictions and simple rules) and the utilization of subsumption to individuate the possible semantic transformations of a query with respect to the schema enriched by integrity constraints.

This paper, however, does not contain a complete formal treatment of the subject; it analizes the feasibility and usefulness of such an approach through some examples.

1.1 An Introductory Example

In order to illustrate our approach, let us consider the following description of part of the organizational structure of a company shown in figure 1, where classes are represented by ellipses, specialization relationships by arrows, base classes are indicated by an asterisk and attributes by oriented arcs (notice that some attributes are not drawn to make the figure more readable).

Employees have a name, they work in a department earning a salary; departments have a name and are managed by a manager; managers are *exactly* those employees who have a level composed of a qualification and a parameter; Employees earn less than the manager of the department in which they work. Technical managers (`TManager`) are *exactly* those employees who have a level composed of a qualification and a parameter greater than 10 and less than 40. Materials are described by a name and a risk; special materials (`SMaterial`) are materials with a feature. Warehouses stock a material, storages are departments and warehouses with a container composed of a kind and a reliability; special storages (`SStorage`) are those storages which stock special materials. The schema of the classes described above (according to a

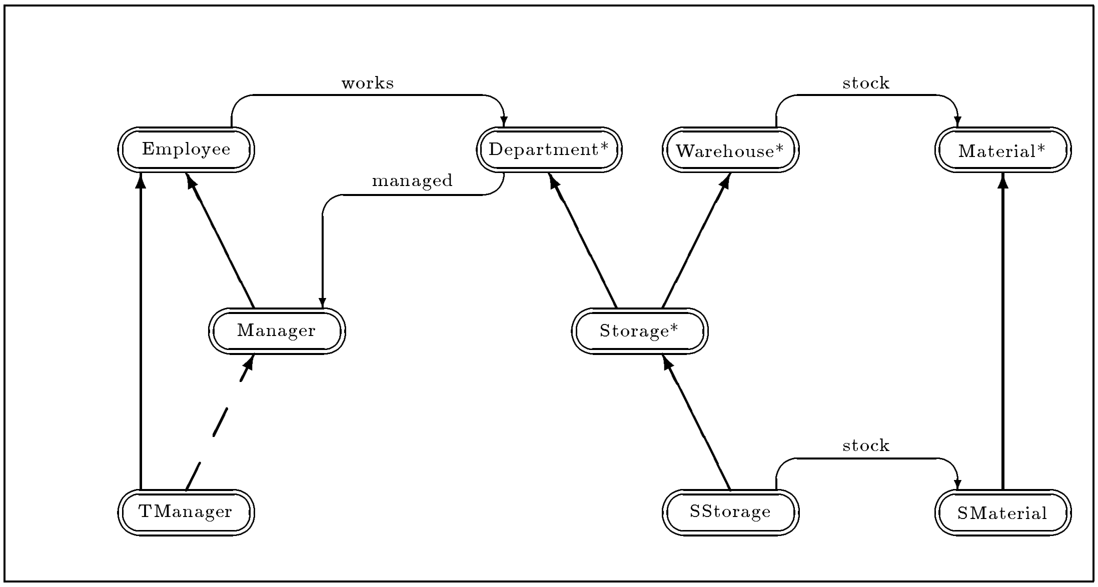

Figure 1: Class Hierarchy

formalism that will be introduced in section 2) is shown in table 1.

It should be noted that, intuitively, descriptions without *exactly* introduce base classes, whereas those with *exactly* introduce derived classes. From the point of view of instantiation, it means that, for example, we have to explicitly fill the base class `Employee`, while every object which is inserted as a employee who has a level composed of a qualification and of a parameter can be automatically inserted in the derived class `Manager`. Moreover, from an intensional point of view, derived classes enable *subsumption relationships* (i.e. specialization relationships implied by class descriptions) to be computed. In figure 1 a computed subsumption relationship between `Manager` and `TManager` is represented by a dashed line.

The language used to specify queries is essentially the same as the one used for defining the database schema, that is, we need to express, in a declarative fashion, the characteristics of the target object, thus defining a new class. The query will be classified in the schema by finding its correct place in the class taxonomy, and the fundamental criterion for this placement is the subsumption relationship between two classes. The first benefit one gains by classifying the query into the class taxonomy will be the validation against the existing schema, to determine if such objects can exist in the database. For example:

Q_1 : Select the TManagers with a parameter less than 8.

$$Q_1 \quad = \quad \texttt{TManager} \sqcap (\triangle.\texttt{level.parameter} < 8)$$

is incoherent, that is it is a contradictory description (it describes a necessarily empty set). In this case, the query can be solved without accessing the database.

Provided the query is semantically well–formed, the classification of the query into the class taxonomy provides us with information that is useful for devising an execution strategy. For example:

Q_2 : Select the Managers with parameter greater than 10 and less than 40.

$$Q_2 \quad = \quad \texttt{Manager} \sqcap (\triangle.\texttt{level.parameter} > 10) \sqcap (\triangle.\texttt{level.parameter} < 40)$$

Table 1: Class Schema

$$
\begin{array}{rcl}
\sigma(\texttt{Level}) & = & [\texttt{qualification}:\texttt{String},\texttt{parameter}:\texttt{Int}] \\
\sigma(\texttt{Employee}) & = & \triangle[\texttt{name}:\texttt{String},\texttt{salary}:\texttt{Int},\texttt{works}:\texttt{Department}] \\
 & & \sqcap\ (\triangle.\texttt{salary} < \triangle.\texttt{works}.\ \triangle\ .\texttt{managed}.\ \triangle\ .\texttt{salary}) \\
\sigma(\texttt{Warehouse}) & = & \triangle[\texttt{stock}:\texttt{Material},\texttt{qty}:\texttt{Int}] \\
\sigma(\texttt{Department}) & = & \triangle[\texttt{name}:\texttt{String},\texttt{managed}:\texttt{Manager}] \\
\sigma(\texttt{Manager}) & = & \texttt{Employee} \sqcap \triangle[\texttt{level}:\texttt{Level}] \\
\sigma(\texttt{TManager}) & = & \texttt{Employee} \sqcap\ \triangle[\texttt{level}:\texttt{Level}] \\
 & & \qquad\qquad \sqcap\ (\triangle.\texttt{level}.\texttt{parameter} > 10) \\
 & & \qquad\qquad \sqcap\ (\triangle.\texttt{level}.\texttt{parameter} < 50) \\
\sigma(\texttt{Storage}) & = & \texttt{Department} \sqcap \texttt{Warehouse} \sqcap [\texttt{container}:[\ \texttt{kind}:\texttt{String}, \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \texttt{reliability}:\texttt{Int}]] \\
\sigma(\texttt{Material}) & = & \triangle[\texttt{name}:\texttt{String},\texttt{risk}:\texttt{Int}] \\
\sigma(\texttt{SMaterial}) & = & \texttt{Material} \sqcap \triangle[\texttt{feature}:\texttt{String}] \\
\sigma(\texttt{SStorage}) & = & \texttt{Storage} \sqcap \triangle[\texttt{stock}:\texttt{SMaterial}]
\end{array}
$$

The query Q_2 will be classified below the class `TManager` (in fact `TManager` subsumes Q_2) and only the objects of such class need be considered to obtain the result. In this case it is mandatory that `TManager` be a derived class.

The location of the query in the class taxonomy can be viewed as a new description of the query, semantically equivalent to the original one, i.e., the classification of the query induces a set of semantic transformations and therefore can be proposed as a means of semantic query optimization. In fact, after the classification, the query Q_2 can be equivalently rewritten as

$$Q_2^{'} \quad = \quad \texttt{TManager} \sqcap (\triangle.\texttt{level}.\texttt{parameter} < 40)$$

In other words, we have obtained a semantically equivalent query using the integrity constraints expressed by class definitions.

In another possible situation, if the query subsumes classes of the schema, then all objects of the subsumed classes are in the solution of the query. Notice that, in this case, the class can be base or derived. For example:

Q_3 : Select the Managers with parameter greater than 10 and less than 60.

$$Q_3 \quad = \quad \texttt{Manager}\ \sqcap (\triangle.\texttt{level}.\texttt{parameter} > 10) \sqcap (\triangle.\texttt{level}.\texttt{parameter} < 60)$$

The above example simply extends to complex object data models the approach followed by [11, 7, 18]. We proposed to further extend this approach by considering a class of integrity constraints represented in the form of simple rules. Let us show how they can be used to devise optimized query execution strategies. For example, consider the following rule and query:

R_1 Materials with risk greater than 20 must be considered as special materials `SMaterial` with a particular feature, namely dangerous.

$$\texttt{SMaterial} \sqcap (\triangle\ \texttt{feature} = \texttt{dangerous}) \quad \leftarrow \quad \texttt{Material} \sqcap (\triangle\ \texttt{risk} > 20)$$

Q_4 : Materials with risk greater than 30.

$$Q_4 \quad = \quad \texttt{Material} \sqcap (\triangle\ \texttt{risk} > 30)$$

Since the query description is subsumed, i.e. *implies*, the antecedent of the rule, the consequent of the rule or a part of it can be added to the query, obtaining a set of semantically equivalent queries. Concerning the example, some of the queries semantically equivalent to Q_4 are the following:

$$\begin{aligned} Q_4^{'} &= \texttt{SMaterial} \sqcap (\triangle\ \texttt{risk} > 30) \\ Q_4^{''} &= \texttt{Material} \sqcap (\triangle\ \texttt{risk} > 30) \sqcap (\triangle\ \texttt{feature} = \texttt{dangerous}) \\ Q_4^{'''} &= \texttt{SMaterial} \sqcap (\triangle\ \texttt{risk} > 30) \sqcap (\triangle\ \texttt{feature} = \texttt{dangerous}) \end{aligned}$$

In $Q_4^{'}$ the search space is limited by a subclass `SMaterial` of the target class `Material`; in $Q_4^{''}$ a factor which can be used in processing the query ($\triangle$ `feature` = `dangerous`) is added; in $Q_4^{'''}$ both the possibilities are considered. Also in this case it does not matter whether the class is base or derived. More complex examples of integrity constraints which, in conjunction with subsumption, can be exploited for query optimization, will be presented in the sequel.

2 A Formalism for Complex Objects including Integrity Constraints

In the following, we will specify the formalism which was introduced in [10]. We call it a formalism as we want it to be the base for describing a number of models that have been proposed.

Types and Values, Classes and Objects

Let $\mathbf{D}$ be the countably infinite set of base-values (which will be denoted by $d, d', \ldots$), e.g., the union of the set of integers, the set of strings, the set of reals, and the booleans. No distinction will be made between base-values and their encoding.

Let $\mathbf{B}$ be a countable set of base-type designators that contains $\mathbf{D}$ (i.e., all mono-valued types), and let $\mathcal{I}_\mathbf{B}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^\mathbf{D}$ such that for all $d \in \mathbf{D}$: $\mathcal{I}_\mathbf{B}[d] = \{d\}$. The special type, denoted by $\bot$, that has an empty interpretation, $\mathcal{I}_\mathbf{B}[\bot] = \emptyset$, will be called *empty type*; this type must be part of $\mathbf{B}$ because the conjunction of different mono-valued types is empty.

We suppose a countable set $\mathbf{A}$ of *attributes* (denoted by $a_1, a_2, \ldots$), a countable set $\mathbf{C}$ of *class-type names* (denoted by $C, C' \ldots$), and a countable set $\mathbf{T}$ of *value-type names* $(T, T', \ldots)$ such that $\mathbf{C}$, $\mathbf{T}$, and $\mathbf{B}$ are pairwise disjoint. Furthermore, $\mathbf{C}$ contains the special symbol $\top_C$.

$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{T})$ denotes the set of all *finite type descriptions* (S, S', ...), also briefly called *types*, over given $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{T}$, that are obtained according to the following abstract syntax rule:

$$S \quad \rightarrow \quad \top \mid B \mid C \mid T \mid \{S\} \mid \langle S \rangle \mid [a_1 : S_1, \ldots, a_k : S_k] \mid S \sqcap S' \mid \ \triangle S$$

$\top$ denotes the *top type* and $\{\}, \langle\rangle, []$ denote the usual type constructors of set, sequence and tuple. The construct $\sqcap$ holds for *conjunction*, whereas $\triangle$, roughly speaking, is an object set forming constructor. By *factor* of a type S we mean one of S_i, if S is of the form $S_1 \sqcap \cdots \sqcap S_n$, $n > 1$, or S otherwise. In the following, we also will write $\mathbf{S}$ instead of $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{T})$ when $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{T}$ are clear from the context.

We assume a countable set $\mathcal{O}$ of *object identifiers* (denoted by $o, o', \ldots$) disjoint from $\mathbf{D}$ and define the set $\mathcal{V}(\mathcal{O})$ of all *values over* $\mathcal{O}$ that are built by the following rule (assuming $p \geq 0$ and $a_1, \ldots, a_p$ distinct attributes in $\mathbf{A}$):

$$v \quad \rightarrow \quad d \mid o \mid \{v_1, \ldots, v_p\} \mid \langle v_1, \ldots, v_p \rangle \mid [a_1 : v_1, \ldots, a_p : v_p]$$

Object identifiers are assigned values by a *total value function* δ from $\mathcal{O}$ to $\mathcal{V}(\mathcal{O})$. In the following we will denote with *domain* the couple $\mathcal{D} = (\mathcal{O}, \delta)$. Given a domain $\mathcal{D}$, an *interpretation function* $\mathcal{I}$ *over* $\mathcal{D}$ is a function from $\mathbf{S}$ to $2^{\mathcal{V}(\mathcal{O})}$ such that:

$$\mathcal{I}[B] = \mathcal{I}_{\mathbf{B}}[B], \quad \mathcal{I}[C] \subseteq \mathcal{O}, \quad \mathcal{I}[T] \subseteq \mathcal{V}(\mathcal{O}) - \mathcal{O}$$

The interpretation of types is defined inductively for all $S, S' \in \mathbf{S}$ by:

$$\begin{aligned}
\mathcal{I}[\{S\}] &= \Big\{\{v_1, \ldots, v_p\} \Big| v_i \in \mathcal{I}[S], 1 \leq i \leq p\Big\} \\
\mathcal{I}[\langle S \rangle] &= \Big\{\langle v_1, \ldots, v_p \rangle \Big| v_i \in \mathcal{I}[S], 1 \leq i \leq p\Big\} \\
\mathcal{I}\Big[[a_1 : S_1, \ldots, a_p : S_p]\Big] &= \Big\{[a_1 : v_1, \ldots, a_q : v_q] \Big| \; p \leq q, v_i \in \mathcal{I}[S_i], 1 \leq i \leq p, \\
&\qquad v_j \in \mathcal{V}(\mathcal{O}), p+1 \leq j \leq q\Big\} \\
\mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S'] \\
\mathcal{I}[\triangle S] &= \Big\{o \in \mathcal{O} \Big| \delta(o) \in \mathcal{I}[S]\Big\} \\
\mathcal{I}[\top] &= \mathcal{V}(\mathcal{O}).
\end{aligned}$$

Atomic Predicates

Another relevant capability of our formalism is *attribute path* support. Attribute paths are sequences of attributes expressing navigation through schema classes and types of complex object data schemata. They are necessary for query languages in OODBMS environment and have been recently introduced to express constraints [14]. In this paper we use attribute paths to express the following types of *atomic predicates*: *range restrictions* on a path terminating on a base type (integer, string, real), and *path equations* to compare two attribute paths. Let us formally the atomic predicates.

Let θ be a *comparison operator*, $\theta \in \{=, \neq, >, <, \geq, \leq\}$, where the elements of the subset $\{>, <, \geq, \leq\}$ are appliable to elements of $\mathbf{D}$, while the operators $\{=, \neq\}$ are appliable to all the elements of $\mathcal{V}$. The equality operator ($=$) has the meaning of *identity*, i.e. two objects are equal if they have the same object–identifier (oid); two tuples, say $v_a = [a_1 : v_1, \ldots, a_p : v_p]$ and $v_b = [a_1^{'} : v_1^{'}, \ldots, a_q^{'} : v_q^{'}]$, are equal if they have the same attributes (i.e. the same *structure*) and equal attribute labels correspond to equal values: $p = q$ and $\forall i$, $i = 1 \ldots p$, $\exists j$, $j = 1 \ldots q$ such that $a_i = a_j^{'}$ and $v_i = v_j^{'}$); two set values, say $v_a = \{v_1, \ldots, v_p\}$ and $v_b = \{v_1^{'}, \ldots, v_q^{'}\}$, are equal if they have the same cardinality and the same elements: $p = q$ and $\forall i$, $i = 1 \ldots p$, $\exists j$, $j = 1 \ldots q$ such that $v_i = v_j^{'}$. For brevity we will not consider the other kind of equality usually defined in complex object data models, namely equality by value.

A attribute path, or simply *path*, c is a sequence of elements $c = e_1 . e_2 . \ldots . e_n$, with $e_i \in \mathbf{A} \cup \{\triangle\}$, $i = 1 \ldots n$, $n \geq 1$. By ϵ we denote a particular path, the *empty path* and by $\mathbf{W}$ we denote the set of all paths. Let c_1 and c_2 be paths, θ a comparison operator and v a base value or an oid ($v \in \mathbf{D} \cup \mathcal{O}$). An atomic predicate P is of one of the forms

$(c_1 \; \theta \; c_2)$ *path equation*
$(c_1 \; \theta \; v)$ *range restriction*

Atomic predicates are types in their own right and are therefore integrated in the type system previously introduced; we add the atomic predicate symbol P to the type syntax described above:

$$S \quad \rightarrow \quad \top \mid B \mid C \mid T \mid \{S\} \mid \langle S \rangle \mid [a_1 : S_1, \ldots, a_k : S_k] \mid S \sqcap S' \mid \ \triangle\, S \mid P$$

In order to define the semantic of a atomic predicate, we introduce, given a domain $\mathcal{D}$, the function $\mathcal{J} : \mathbf{W} \longrightarrow 2^{\mathcal{V} \times \mathcal{V}}$, associating a set of value pairs to every path:

- empty path: $\mathcal{J}[\epsilon] = \Big\{ (v_1, v_1) \in \mathcal{V} \times \mathcal{V} \Big\}$
- single element path: $\mathcal{J}[\triangle] = \Big\{ (o, v) \in \mathcal{O} \times \mathcal{V} \mid \delta(o) = v \Big\}$
  $\mathcal{J}[a] = \Big\{ (v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid v_1 = [\ldots, a : v_2, \ldots] \Big\}$
- multiple element path: $\mathcal{J}[e_1 . e_2 . \ldots . e_n] = \mathcal{J}[e_1] \circ \mathcal{J}[e_2] \circ \ldots \circ \mathcal{J}[e_n]$
  where $\circ$ is the symbol of function composition.

Now we can extend the interpretation function $\mathcal{I}$ to atomic predicates, as follows:

$$\begin{aligned}
\mathcal{I}[(c_1\ \theta\ v')] &= \Big\{ v \in \mathcal{V} \mid \text{ there exists } v' \in \mathcal{V} \text{ such that } \\
&\qquad (v, v') \in \mathcal{J}[c_1], \text{ and } (v' \theta v) \text{ is true} \Big\} \\
\mathcal{I}[(c_1\ \theta\ c_2)] &= \Big\{ v \in \mathcal{V} \mid \text{ there exists } v_1, v_2 \in \mathcal{V} \text{ such that } \\
&\qquad (v, v_1) \in \mathcal{J}[c_1], (v, v_2) \in \mathcal{J}[c_2], \text{ and } (v_1\ \theta\ v_2) \text{ is true} \Big\}
\end{aligned}$$

Database Schema

Given a base-type system $\mathbf{B}$ and finite sets of attributes $\mathbf{A}$, class names $\mathbf{C}$, and value-type names $\mathbf{T}$, a *schema* σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{T})$ is a total function $\sigma : \mathbf{C} \cup \mathbf{T} \longrightarrow \mathbf{S}$.

Let $\mathbf{C_b}$ and $\mathbf{C_d}$ be a partitioning of $\mathbf{C}$. $\mathbf{C_b}$ is called the set of *base classes* $(P, P', \ldots)$ and $\mathbf{C_d}$ the set of *derived classes* $(D, D', \ldots)$. The interpretation of the former has to be provided by the user, while the interpretation of the latter is derived (hence the name) from its definition and the interpretation of base classes, corresponding to a view (i.e. a query) in a database context. In particular, $\mathbf{C_d}$ contains the $\top_C$ class and $\sigma(\top_C) = \triangle\top$.

We will require some well-formedness conditions on a schema, namely, *well-foundedness of value-type definitions* and of *the inheritance relation*. We say that a schema σ is *type well-founded* if types defined using other type names always describe finitely nested values.

As our model supports only structural descriptions of class and value type, we deal with structural inheritance, usually strict and denoted as *isa* relationships in DB environment. The inheritance is expressed by conjunction ($\sqcap$) in a type or class definition. Formally, we say that N, $N \in \mathbf{C} \cup \mathbf{T}$, *inherits directly from* N', written $N \prec\!\!\!\cdot\, N'$, iff $\sigma(N) = N' \sqcap S$. A schema is *inheritance well-founded* iff the transitive closure of $\prec\!\!\!\cdot$, which is denoted by $\prec$, is a strict partial order.[1] The reflexive and transitive closure will be denoted by $\preceq$. Type well-founded and inheritance well-founded schemata are called *well-formed*.

Let us briefly comment on the schema shown in table 1. First of all, the $\triangle$ operator: it permits to give *class descriptions*, i.e. types whose interpretation is a set of object identifiers. Descriptions without $\triangle$ are related to value-type names (as in the case of `Level`). Secondly,

[1]Although this restriction is not necessary from a purely technical point of view, it is a restriction which ensures that the intuitive meaning of inheritance is satisfied.

the conjunction operator: it allows the explicit representation of *simple inheritance*, as in the case of the class `Manager` (which inherits from the class `Employee`) or *multiple inheritance*, as in the case of the class `Storage` (which inherits from the classes `Department` and `Warehouse`). Finally, we show the meaning of the symbols used in building a path and the semantics of atomic predicates. Consider the predicate ($\triangle$ `level parameter` $>$ 10) related to the class `TManager`. The path $\triangle$ `level parameter` navigates through the `TManager` structure: by $\triangle$ ($e_1 = \triangle$) the values associated to the `TManager` objects, which are tuples, are addressed and by `level` ($e_2 =$ `level`) and `parameter` ($e_3 =$ `parameter`) the specific values of this fields are addressed.

Database Instance

In the following, we will specify the kinds of structures that are are described by our schemata—by defining the *legal instances* of a schema.

We say that an interpretation function $\mathcal{I}$ as defined above is a *possible instance* of a schema σ iff the set $\mathcal{O}$ is *finite*, and for all $P \in \mathbf{C_b}, D \in \mathbf{C_d}, T \in \mathbf{T}$:

$$\mathcal{I}[P] \subseteq \mathcal{I}[\sigma(P)], \quad \mathcal{I}[T] = \mathcal{I}[\sigma(T)], \quad \mathcal{I}[D] = \mathcal{I}[\sigma(D)].^2$$

As a first observation, a schema is usually given only on the basis of P and T and thus its possible instance is defined by the first two formulae. The interpretation of base classes P is explicitly given by the user and the interpretation of type names T can be uniquely computed as type names describe always finitely nested values.

In our case, we add to the schema derived classes D, whose interpretation must be computed on the basis of their descriptions. The presence of cyclic derived classes [3] gives raise to the problems for the computation of their interpretation, as has been shown for terminological languages in [5, 17] and in [8, 9] for complex object data models. In this paper we assume that cyclic derived classes do not occur in schemas.

Subsumption

Let us define the *subsumption relation* of a schema σ:

$$S \leq S' \quad \text{iff} \quad \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ for all } \mathcal{D} \text{ and for all legal instances } \mathcal{I} \text{ of } \sigma.$$

It follows immediately that $\leq$ is a preorder (i.e., transitive and reflexive but antisymmetric) that induces an *equivalence relation* $\simeq$ on types: $S \simeq S'$ iff $S \leq S'$ and $S' \leq S$. Moreover, we say that a type S is *incoherent* iff $S \simeq \bot$. This is equivalent to saying that for all domain $\mathcal{D}$ the type extension is always empty. A schema σ is called *coherent* if it does not contain any incoherent type or class name[4].

Let us briefly comment the relationship existing between the inheritance and subsumption relations; in a well-formed schema, a class C that inherits from another class C', i.e., $C \prec C'$, is always interpreted as a subset of the interpretation of C': conseguently, if $C \prec C'$ then $C \leq C'$. The converse, in general, does not hold, the main reason being the presence of derived classes. Infact we have seen that `TManager` $\leq$ `Manager`, even if `TManager` $\not\prec$ `Manager`. Another reason, more specific, is the presence of incoherent classes, since an incoherent class is subsumed by every other class. Finally, in a well-formed coherent schema σ we can state a result that is a partial converse of the last one: for all $C \in \mathbf{C}$ and all $P \in \mathbf{C_b}$, $C \leq P$ if an only if $C \prec P$. In

[2] Note that $\mathcal{I}[\top_C] = \mathcal{I}[\triangle\top] = \mathcal{O}$.

[3] A class name C is *cyclic* iff its description refers directly or indirectly to C itself.

[4] Obviously there may be incoherent types used in set and sequence types.

other words, a base class P subsumes another class C if and only if C explicitly inherits from P. In the following only well-formed coherent schemas will be considered.

Concerning subsumption computation in presence atomic predicates we note the following. In [10] it has been shown that atomic predicates of the form $(c_1\ \theta\ v)$ can be represented in cyclic terminologies by means of base type intervals without introducing complexity problems in subsumption computation. In [6] a schema is proposed to integrate in terminological languages the so called *concrete domains* (which are a superset of the base types in our model) and a set of predicates typical of such domains. In the same paper it has been proved that subsumption in this context, considering only acyclic terminologies, is still decidable.

Rules

In this section we introduce universally quantified *if then* rule over the objects of the domain to express integrity constraints:

> **for all** $x \in \mathcal{O}$
> **if** x is of type R^a
> **then** x must be of type R^c

where R^c and R^a represent respectively the *consequent* and the *antecedent* of the rule and are type expressions of the σ extended with atomic predicates.

As a rule expresses an integrity constraint which must be satisfied by each database state its semantics is informally the following:

> for any given object **if** the value associated with the object **verifies** a given description (antecedent of the rule), **then** that value **must verify** other conditions (consequent of the rule).

Given a schema σ, a rule R on σ is expressed in the following form:

$$R^c \quad \leftarrow \quad R^a$$

where R^c and R^a are type expressions relative to σ. Given a domain $\mathcal{D}$, we say that an instance $\mathcal{I}$ of σ on $\mathcal{D}$ *satisfies* a rule $R = R^c \ \leftarrow \ R^a$ if

$$\mathcal{I}[R^a] \subseteq \mathcal{I}[R^c]$$

Given a schema σ, a domain $\mathcal{D}$ and a set of rules on σ (denoted by $\mathbf{R}$), we say that an instance $\mathcal{I}$ of σ on $\mathcal{D}$ is *legal w.r.t. the rules* if $\mathcal{I}$ satisfy all the rules of $\mathbf{R}$. In the following, a legal instance w.r.t. the rules will be denoted by $\mathcal{I}_R$.

In general we have, for each type S of σ, $\mathcal{I}_R[S] \subseteq \mathcal{I}[S]$, that is rules restrict the set of legal instances of a database schema. Given a rule $R = R^c \ \leftarrow \ R^a$ consider $\mathcal{I}_R$ with $\mathbf{R} = \{R\}$, if the antecedent is subsumed by the consequent then $\mathcal{I}_R[S] = \mathcal{I}[S]$, that is, the rule is obviously redundant.

In the following we give some rules related to previously introduced classes.

R_2**:** Storages storing a quantity of material greater than 50 must be managed by a specialized manager `TManager`.

$$\triangle\Big[\texttt{managed:TManager}\Big] \quad \leftarrow \quad \texttt{Storage} \sqcap (\triangle.\texttt{qty} \ > 50)$$

R_3**:** Storages storing materials having a risk greater than 20 must be `SStorage` with a container reliability greater than 5.

$$\texttt{SStorage} \sqcap (\triangle.\texttt{container.reliability} > 5) \quad \leftarrow \quad \texttt{Storage} \sqcap (\triangle.\texttt{stock.} \triangle \texttt{.risk} > 20)$$

3 Semantic Query Optimization by Subsumption

Let us give some definitions in order to introduce the subject of semantic query optimization. Informally, two queries are semantically equivalent if they return the same answer for any database state. In database environment, integrity constraints expressed as rules, constitute the basic tool for performing semantic query optimization [13, 19, 20]. In artificial intelligence environment, the approach followed for semantic query optimization is to exploit *classification*. A query is viewed as a derived class and by computing all subsumption relations between it and the class taxonomy, it is possible to classify it, that is, to individuate its right position in the taxonomy [7, 11, 18].

Our proposal is to couple and enhance the above mentioned approaches as we perform semantic query optimization by exploiting both classification and integrity constraints to obtain semantically equivalent queries.

In our framework, including integrity constraints as rules, we can give the following definition of semantically equivalent queries: given a schema σ and a set of rules R, Q and Q' are semantically equivalent, denoted as $Q \equiv Q'$, if and only if $\mathcal{I}_R[Q] = \mathcal{I}_R[Q']$ for all domains $\mathcal{D}$ and for all interpretation functions $\mathcal{I}_R$ of σ on $\mathcal{D}$.

The proposed method exploits subsumption for classifying a query in a class hierarchy including (besides classes) the antecedents and the consequents of the integrity constraint rules given on the schema; in the following we describe by simple examples the main features of our method.

First of all let us show the applicability of classification, as proposed in artificial intelligence environment, in query processing.

As a result of the classification of a query, either an equivalent class or the set of immediate subsumers and immediate subsumees classes is returned. In the former case, the answer is simply the set of all objects that are instances of the class equivalent to the query. In the latter case, the union of the sets of instances of the immediate subclasses are included in the answer (that is do not need to access instances to check the query) and only objects that are instances of immediate superclasses but not instances of the immediate subclasses have to be checked against the query.

Notice that this technique is heavily enhanced by the presence of derived classes. In fact, if the schema contains only base classes, the set of immediate subsumers of the query coincides with the set of superclasses explicitly given in the query description. On the contrary, if the schema contains also derived classes the set of immediate subsumers of the query can be different from the superclasses explicitly given, i.e., more precisely, it is a specialization of the explicitely given superclasses.

We can improve classification effectiveness for schemata containing only base classes by using the integrity constraint rules introduced in our framework. In fact, by exploiting rules we can individuate other classes (eventually base classes) whose interpretation includes the answer. The most significant situation occurs when the query is subsumed by the antecedent of a rule and the consequent of that rule is of the form $C' \sqcap \ldots$; for rules semantics, the query is a subset of the C' interpretation.

Let us introduce as an example the query Q_5:

Q_5 : Select storages storing materials having a risk greater than 40

$$Q_5 \quad = \quad \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\triangle.\texttt{risk} > 40)$$

as the query is subsumed by the antecedent of rule R_3 and the consequent is of the form $\texttt{SStorage} \sqcap \ldots$, we can operate the transformation:

$$Q_5^{'} \quad = \quad \texttt{SStorage} \sqcap (\triangle.\texttt{stock}.\triangle.\texttt{risk} > 40)$$

which limits the search space at a subclass `SStorage` of the explicitly given class `Storage`.

Following the approach of semantic query optimization works, [19, 20] in general, we can perform query transformations by **constraint introduction**: If the query is subsumed by the antecedent of a rule then the consequent of that rule can be added to the query. Thus, for example, we can transform Q_5 into Q_5'':

$$Q_5'' \quad = \quad \texttt{SStorage} \sqcap (\triangle.\texttt{container.reliability} > 5) \sqcap \ (\triangle.\texttt{stock.} \triangle.\texttt{risk} > 40)$$

Following [19, 20], the decision of using or not the added factor in the query execution is taken by means of heuristics which takes into account the presence of data index (*index introduction*) and of the query execution plan (*scan reduction*).

Furthermore, we can limit the search space of an attribute of a query by substituting in a query a subclass of an explicitly given class when useful rules are available. For example:

Q_6 : Select storages storing materials having a risk greater than 50 and a quantity greater than 100 and managed by a manager with a salary less than 50000

$$\begin{aligned} Q_6 \quad = \quad \texttt{Storage} \ & \sqcap (\triangle.\texttt{stock.} \triangle.\texttt{risk} > 50) \\ & \sqcap (\triangle.\texttt{qty} \ > 100) \\ & \sqcap (\triangle.\texttt{managed.} \triangle.\texttt{salary} < 50000) \end{aligned}$$

whose classification in the schema including rules is shown in figure 2. In this case two rules have been used for the query transformation:

$$\begin{aligned} Q_6' \quad = \quad \texttt{SStorage} \ & \sqcap [\texttt{managed:TManager} \sqcap (\triangle.\texttt{salary} < 50000)] \\ & \sqcap (\triangle.\texttt{stock.} \triangle.\texttt{risk} > 50) \\ & \sqcap (\triangle.\texttt{qty} > 100) \end{aligned}$$

As in the previous query, the search space has been constrained to the subclass `SStorage` of the explicitly given class `Storage` by rule R_3; moreover, by rule R_2 the search space for the attribute `managed` has been limited to a subclass `TManager` of the class `Manager` on which `managed` is explicitly defined.

By rules we can individuate, besides the immediate subclasses computed by subsumption, further classes whose objects can be included in the answer set of the query. In fact, if the query subsumes the consequent of a rule, then all objects of the antecedent of that rule (and then all objects of the classes subsumed by that antecedent) are in the query solution. For example, if we introduce in the schema the following class:

$$\sigma(\texttt{PStorage}) \quad = \quad \texttt{Storage} \sqcap (\triangle.\texttt{stock.} \triangle.\texttt{risk} > 30)$$

and the query is:

Q_7 : Select storages with a container reliability greater than 2

$$Q_7 \quad = \quad \texttt{Storage} \sqcap \ (\triangle.\texttt{container.reliability} > 2)$$

Query Q_7 subsumes the consequent of the rule R_3 and as the antecedent of that rule subsumes the class `PStorage`, then all objects of `PStorage` are in the solution of Q_7.

Another rewriting rule for query optimization proposed in [19, 20] is the **constraint removal**: If the query is subsumed by the antecedent of a rule and the consequent of that rule is subsumed by the conjunction of some factors of the query then these factors may then be removed from the query.

For example:

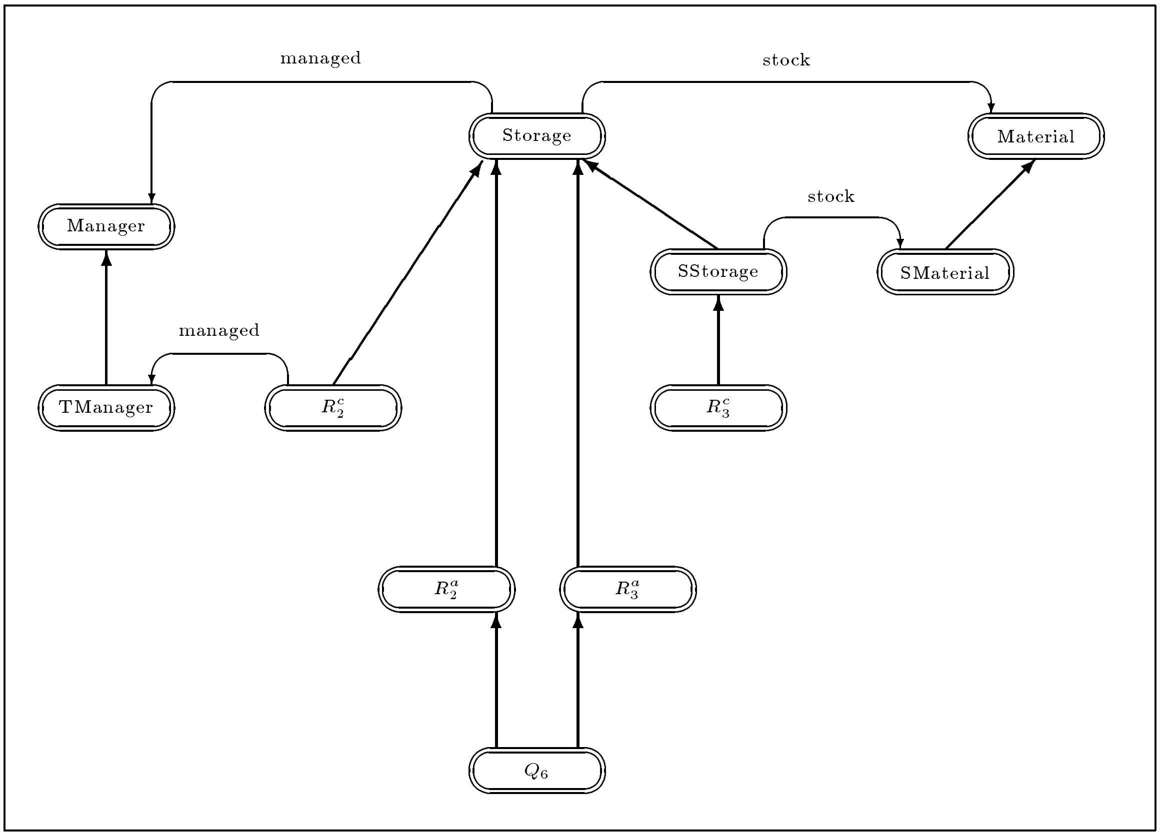

Figure 2: Classification of the query Q_6

Q_8 : Storages storing a quantity of materials greater than 100, and managed by a manager with parameter between 10 and 60

$$\begin{aligned} Q_8 \quad = \quad & \texttt{Storage} \sqcap (\triangle.\texttt{qty} > 100) \\ & \sqcap (\triangle.\texttt{managed.}\ \triangle\ \texttt{.level.parameter} > 10) \\ & \sqcap (\triangle.\texttt{managed.}\ \triangle\ \texttt{.level.parameter} < 60) \end{aligned}$$

The query is subsumed by the antecedent of the rule R_2 and the conjunction of the factors $(\triangle.\texttt{managed.}\ \triangle\ \texttt{.level.parameter} > 10)$ and $(\triangle.\texttt{managed.}\ \triangle\ \texttt{.level.parameter} < 60)$ subsumes the consequent of the same rule. These factors may therefore be deleted from Q_8, then obtaining the semantically equivalent query $Q_8^{'}$:

$$Q_8^{'} \quad = \quad \texttt{Storage} \sqcap (\triangle.\texttt{qty} > 100)$$

In this case again we apply heuristics to decide about the elimination or not of the query factors. In particular, notice that the elimination of a factor which expresses navigation through schema classes avoids to access a those classes (in the example, we avoid to access to the class `Manager`).

4 Conclusions

The crucial point of semantic query optimization is the choice of the most favorable, among the many possible, query transformations. In the present work we have not dealt with the quantification of the utility of a query transformation, but we provided only an intuitive justification of some typical transformations. In particular, we assumed that the reference to a more specific class is always favorable, since it requires to access to a smaller amount of data. The research will continue towards a true optimization strategy either by finding a set of heuristic transformation rules or by examining exhaustively all the possible transformations. In any case, the optimization shall take into account the various possible storage models.

References

[1] S. Abiteboul and S. Grumbach. Col: a logic-based language for complex objects. In S. Ceri, J.W. Schmidt, and M. Missikoff, editors, *EDBT '88 - Lecture Notes in Computer Science N.303*, pages 271–293. Springer-Verlag, 1988.

[2] S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems*, 12(4):525–565, 1987.

[3] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[4] P. Atzeni, F. Cacace, S. Ceri, and L. Tanca. The LOGIDATA$^+$ model. Technical Report 5/20, CNR - Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, 1990.

[5] F. Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *8th Int. Conf.*, Boston, Mass., USA, 1990.

[6] F. Baader and P. Hanscgke. A scheme for integrating concrete domains into concept languages. Technical report, Kaiserslautern, Germany, 1990.

[7] H.W. Beck, S.K. Gala, and S.B. Navathe. Classification as a query processing technique in the CANDIDE data model. In *5th Int. Conf. on Data Engineering*, pages 572–581, Los Angeles, CA, 1989.

[8] C. Beeri. A formal approach to object-oriented databases. *Data and Knowledge Engineering*, 5:353–382, 1990.

[9] D. Beneventano and S. Bergamaschi. Subsumption for complex object data models. In J. Biskup and R. Hull, editors, *4th Int. Conf. on Database Theory - Berlin*, pages 357–375, Heidelberg, Germany, October 1992. Springer-Verlag.

[10] S. Bergamaschi and B. Nebel. Theoretical foundations of complex object data models. Technical Report 5/91, Roma, January 1992.

[11] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.

[12] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[13] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[14] N. Coburn and G.E. Weddel. Path constraints for graph-based data models: Towards a unified theory of typing constraints, equations and functional dependencies. In *2nd Int. Conf. on Deductive and Object-Oriented Databases*, pages 312–331, Heidelberg, Germany, December 1991. Springer-Verlag.

[15] C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, 1989.

[16] C. Lécluse and P. Richard. The O_2 data model. In *Int. Conf. On Very Large Data Bases*, 1989.

[17] B. Nebel. *Terminological Cycles: Semantics and Computational Properties*, chapter 11, pages 331–362. Morgan Kaufmann Publishers, Inc., 1991.

[18] B. Nebel and C. Peltason. Terminological reasoning and information management. Technical report, Universita' di Berlino, 1991.

[19] S. Shenoy and M. Ozsoyoglu. A system for semantic query optimization. *ACM-SIGMOD*, pages 181–195, May 1987.

[20] M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimization. *ACM Transactions on Database Systems*, 17(4):563–600, December 1992.