Semantic Query Optimization by Subsumption in OODB*

Domenico Beneventano◇, Sonia Bergamaschi◇, Claudio Sartori°

Int. Workshop on Flexible Query-Answering Systems (FQAS96), Roskilde, Denmark, may 1996

Abstract: The purpose of semantic query optimization is to use semantic knowledge (e.g. integrity constraints) for transforming a query into a form that may be answered more efficiently than the original version. This paper proposes a general method for semantic query optimization in the framework of Object Oriented Database Systems. The method is applicable to the class of conjunctive queries and is based on two ingredients: a formalism able to express both class descriptions and integrity constraints rules as types; *subsumption* computation between types to evaluate the logical implications expressed by integrity constraints rules.

1 Introduction

The purpose of semantic query optimization is to transform a query into an *equivalent* one that may be answered more efficiently than the original version. The basic idea is to use semantic knowledge about the database to transform the query. Informally, *semantic equivalence* means that the transformed query has the same answer as the original query on all databases satisfying the integrity constraints. The notion of semantic query optimization for relational databases was introduced in the early 80's by King [17, 18]; Hammer and Zdonik [13] independently developed very similar optimization methods. The key idea in [17, 18], as well as in [13], is that integrity constraints may not only be utilized to enforce consistency of a database but also to optimize user queries. During the last decade, many efforts have been made to improve this technique and to generalize it to deductive databases (Shenoy and Ozsoyoglu [21, 22], Siegel et al. [23], Chakravarthy et al. [10]). Deductive databases provide a theory of relational databases as well as a query language which is more powerful than relational query languages. Furthermore, they provide an uniform and powerful representation (in terms of clause form) for expressing database components (facts, deductive rules, as well as integrity constraints) and the necessary formalism (in term of satisfiability, provability and inference mechanism) for developing a theory of semantic query optimization. More recently, some efforts have been made to perform semantic query optimization in Object Oriented Database Systems (OODBs) [11, 14, 20, 8]. The main point is that OODBs provide a very rich type (class) system able to directly represent a subclass of integrity constraints in the database schema. By exploiting schema information as, for instance inheritance relations between types (classes), it is possible to perform semantic query optimization.

*This research has been partially funded by the Basi di Date Evolute - MURST 40% project.

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

◇ Dipartimento di Scienze dell'Ingegneria, Università di Modena, Via G. Campi 213/A, I-41100 Modena, Italy.

e_mail: {dbeneventano, sbergamaschi, csartori}@deis.unibo.it

This work represents an improvement and a generalization of the above works for many aspects. Our method applies to OODBs and, with respect to previous proposal in this environment, we extend the type system including a large class of integrity constraints. Furthermore, we provide an uniform and powerful representation formalism (in terms of types) for expressing database components (classes, as well as integrity constraints) and queries. Finally, we provide the theoretical framework (in term of *subsumption*) for developing a theory of semantic query optimization which includes the different query transformation criteria proposed in the literature.

Since query languages for OODBs are more expressive than our formalism we, following [8], ideally introduce a separation of a query into a *clean* part that can be represented as a type in our formalism and a *dirty* part that goes beyond the type system expressiveness. Semantic optimization will be performed only over the clean part of a given query. The clean part of a query, in the following referenced as query, corresponds to so–called conjunctive queries or single operand queries [16] in OODBs and is an object–type in our formalism. A query is a *virtual class*, that is a class whose description represents necessary and sufficient conditions, whereas database classes represent only necessary condition.

The chosen strategy for optimization is the following. Prior to the evaluation of any query, we *compile*, once at all, the database schema (classes + integrity constraints rules), giving raise to a new schema which is enriched by (*all the*) *logically implied isa relationships.*

The compilation process is based on the generation of the *semantic expansion* in *canonical form* (i.e. a form which permits to abstract from different syntactical representation of semantically equivalent types) of the schema types. Following the approach of [21, 23] for semantic query expansion, the semantic expansion of a type permits to incorporate any possible restriction which is not present in the original type but is *logically implied* by the type and by the schema. Semantic expansion is based on the iteration of this simple transformation: if a type implies the antecedent of an integrity constraint rule then the consequent of that rule can be added. Logical implications are evaluated by means of the *subsumption computation* [7, 5, 4] between types.

At run time, we add to the compiled schema the query and activate the compilation process again for the query: its semantic expansion, possibly including new *isa* relationships logically implied by the integrity constraints rules, is obtained. If new *isa* relationships are found, it is possible *to move the query down* in the schema hierarchy: this is the main optimization achievement of the proposed method.

Due to space limitations, we cannot adequately compare our work with the related ones, but we refer to the classification proposed by Chakravarthy et al. [10], to further motivate it: "There are several aspects to semantic query optimization: the type of database under consideration and the type of integrity constraints allowed; the generation of semantically equivalent queries and their correctness; filtering of unuseful information; the generation process of semantically equivalent queries control and its correctness; the integration of semantic query optimization with conventional query optimization." With reference to the first aspect, we restrict integrity rules to the ones expressible by logical implications between types; in this way we can efficiently compute implications by a subsumption algorithm. With reference to the subsequent ones, we deal with:

- *Correctness and semantic equivalence*: the semantic expansion of a query is a semantically equivalent query; this result follows from the formal semantics of types, subsumption relationship and semantic

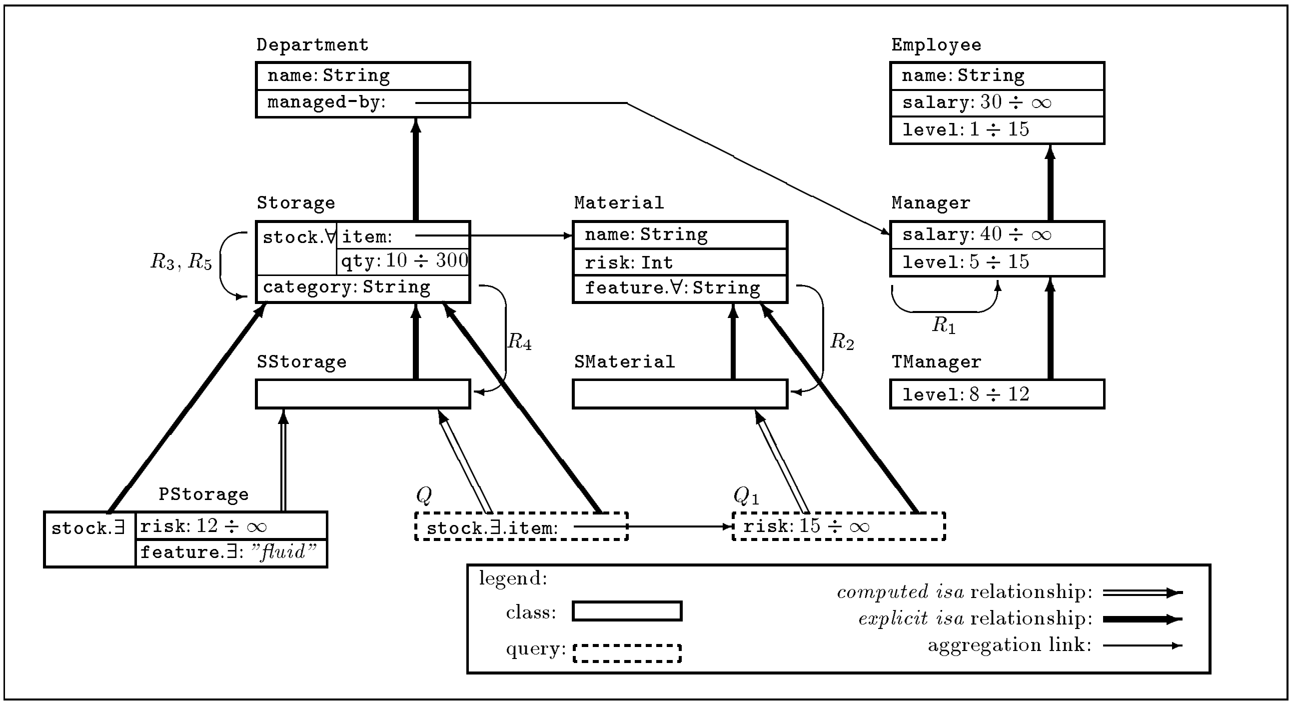


Figure 1: The Company domain with two optimized queries

expansion; from the correctness of the subsumption algorithm and from the *extension preserving* property of the canonical form transformation.

- *Controlling the generation process and filtering*: the semantic expansion of a query is the *most specialized query* among the equivalent queries (it follows from the completeness of the subsumption algorithm). In particular, during the transformation, we compute and substitute in the query, at each step, the *most specialized classes* satisfying the query. A further filtering activity can be performed as we are able to detect the *eliminable* factors of a query, that is, the factors logically implied by the query.

The generality and independence from any specific cost model and storage details of the proposed method makes it suitable for an easy integration with a traditional query optimizer.

In order to illustrate our method, let us introduce the Company domain example.

The Company Domain Example

The Company domain is part of the organizational structure of a company (see table 1 for its formal description and figure 1 for its graphical representation): Employees have a name and a salary (greater equal to 30K) and a level (1 to 15). Managers are employees with high levels (5 to 15) and high salaries (greater equal to 40K); tmanagers are managers with a subrange of level values (8 to 12). Departments have a name and are managed by managers. Materials have a name, a risk and a set of features; smaterials are a subset of materials. Storages are departments with a category property and stocking a set of materials of a given quantity; sstorages are a subset of storages. Pstorages are subset of storages, characterized by stocking at least a material with a *"fluid"* feature and a risk value greater equal to 12. Rule R_1 says that managers with a level from 5 to 10 must have a salary from 40K to 60K. R_2 says that, having a risk greater equal to 10, is a sufficient condition for a material to be an smaterial (note that it is different from giving for smaterial the following description: $\texttt{Material} \sqcap \triangle[\texttt{risk}: 10 \div \infty]$, as this last description expresses a necessary condition whereas R_2 expresses a sufficient condition). R_3, R_4 and R_5 are rules on storages: R_3 constrains storages, stocking at least a material with a *"F1"* feature, to be managed by a tmanager; R_4 says that, stocking at least an smaterial is a sufficient condition for a storage to be an sstorage; R_5 says that, if a storage stocks at least a material with a quantity from 10 to 50, it is of category "A2".

Figure 1 shows the schema enriched by a *computed isa* relationship between the classes `PStorage` and `SStorage` and the semantic optimization with of a query Q: "Select storages storing *at least* a material having a risk greater equal to 15" In figure 1, classes are denoted by boxes with slots indicating properties; inheritance relationships (*explicit isa*) are denoted by thick arrows, aggregation links by thin arrows and rules by arcs; queries are drawn with dashed box. Double arrows correspond to *computed isa* relationship, logically implied by the classes and the rules of the database schema. Let us rewrite query Q as: "Select storages storing *at least* an element of Q_1", where Q_1:" Select materials having a risk greater equal to 15". After semantic optimization we obtain the computed positions for the queries Q, Q_1 in the hierarchy (drawn by double arrows). As the queries have been pushed down in the hierarchy, we have an obvious optimization as the set of objects satisfying the query has been restricted to the subclass `SStorage` (`SMaterial`).

The outline of the paper is as follows. Section 2 introduces the formalism syntax and semantics; in particular subsection 2.3 introduces the subsumption relationship (definition and algorithm) and the semantic expansion function. In Section 3, the method effectiveness is shown by some examples and by detailing the steps for achieving the semantic optimization of the queries of figure 1. The appendix includes: the canonical transformation function; some results on coherence preservation of a database schema with rules.

2 A Formalism for Complex Objects and Integrity Constraints

The presented formalism is an extension of the *object description language* ODL, introduced in [4] and which basically follows complex object data models [1, 19].

The main features of ODL are described in the following. In ODL a rich structure for the *system of atomic types*, the system of non-decomposable, basic value-types is assumed. Besides the atomic types *integer*, *boolean*, *string*, *real*, and *mono-valued types*, ODL consider also the possibility that subsets of these types are used, e.g., intervals of integers. The only restriction is that the system of atomic types is closed w.r.t. intersection and that the intersection of two atomic types can be computed in polynomial time.

Based on these atomic types, *tuple*, *set*, and *class types* can be created. Class types (also briefly called *classes*) denote sets of *objects with an identity and a value*, while the former two types – also called *value types* denote sets of *complex, finitely nested values without object identity*. Additionally, a conjunction operator can be used to create intersections of previously introduced types. This operator can be used to specify simple and multiple inheritance. For value types, no circular references are permitted in order to guarantee that values are always only finitely nested. Class types, on the other hand, can be defined making circular references.

The extensions to ODL introduced in the present work are: *quantified path types* and *integrity constraint rules*. The former extension has been introduced to deal easily and powerfully with nested structures. Paths, which are essentially sequences of attributes, represent the central ingredient of O–O query languages to navigate through the aggregation hierarchies of classes and types of a schema [15, 6]. In particular, following [6], we provide *quantified* paths to navigate through set types. The allowed quantifications are existential and universal and they can appear more than once in the same path. A path type is a type associating a path to a type of the formalism. Thus, by means of path types, as recently introduced in [12], we can express a class of integrity constraints.

The latter extension allows the expression of integrity constraints represented as a *if then rule* universally quantified over the elements of the domain with an antecedent and a consequent which are types of the formalism. These rules permit to represent a relevant piece of knowledge in a declarative style.

2.1 Database Schema and Rules

Let $\mathbf{D}$ be the countably infinite set of base-values (which will be denoted by $d_1, d_2, \ldots$), e.g., the union of the set of integers, the set of strings, and the booleans.

Let $\mathbf{B}$ be a countable set of base-type designators that contains $\mathbf{D}$ and let $\mathcal{I}_\mathbf{B}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^\mathbf{D}$ such that for all $d \in \mathbf{B}$: $\mathcal{I}_\mathbf{B}[d] = \{d\}$. Let "$\sqcap$" be an operation on $\mathbf{B}$ defined by: $B' \sqcap B'' = B$ iff $\mathcal{I}_\mathbf{B}[B'] \cap \mathcal{I}_\mathbf{B}[B''] = \mathcal{I}_\mathbf{B}[B]$. We say that $\mathbf{B}$ is a *base-type system* iff $\mathbf{B}$ is complete with respect to $\sqcap$. The special type that has an empty interpretation will be called *empty type* and is denoted by $\bot$.[1] We will say that $\mathbf{B}$ is a *PTIME base-type system* iff $B' \sqcap B'' = B$ can be decided in polynomial time. Furthermore if for each subset $\mathbf{X} \subseteq \mathbf{B}$ with $\bigsqcap \mathbf{X} = B$, there are two elements $B', B'' \in \mathbf{X}$ such that $B' \sqcap B'' = B$, $\mathbf{B}$ is a *binary compact* base-type systems. In the following we well adopt as base-type system $\mathbf{B} = \{\mathtt{Int}, \mathtt{String}, \mathtt{Bool}, \mathtt{Real}, i_1\text{–}j_1, i_2\text{–}j_2, \ldots, d_1, d_2, \ldots\}$, where the d_k's denote all the elements of $\mathtt{Int} \cup \mathtt{String} \cup \mathtt{Bool}$ and the $i_k\text{–}j_k$'s denote all possible ranges of integers (i_k can be $-\infty$ to denote the minimum element of $\mathtt{Int}$ and j_k can be $+\infty$ to denote the maximum element of $\mathtt{Int}$). Assuming the standard interpretation of the base-type designators, $\mathbf{B}$ is obviously a binary compact base-type system.

Let $\mathbf{A}$ be a countable set of *attributes* (denoted by $a_1, a_2, \ldots$) and $\mathcal{O}$ a countable set of *object identifiers* (denoted by $o, o', \ldots$) disjoint from $\mathcal{D}$, 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 $\mathbf{A}$):

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

Object identifiers are assigned values by a *total value function* δ from $\mathcal{O}$ to $\mathcal{V}(\mathcal{O})$.

Let $\mathbf{C}$ be a countable set of names for *base class* (denoted by $C, C' \ldots$), such that $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are pairwise disjoint. A *path* p is a sequence of elements $p = e_1 . e_2 . \ldots . e_n$, with $e_i \in \mathbf{A} \cup \{\triangle\} \cup \{\forall\} \cup \{\exists\}$. We denote with ϵ a particular path, the *empty path*. $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ denotes the set of all *finite type descriptions* (S, S', ...), also briefly called *types*, over given $\mathbf{A}, \mathbf{B}, \mathbf{C}$, obtained according to the following syntax rule:

$$S \rightarrow \top \mid B \mid C \mid [a_1 : S_1, \ldots, a_k : S_k] \mid \forall\{S\} \mid \exists\{S\} \mid \triangle S \mid S \sqcap S' \mid (p : S)$$

$\top$ denotes the *top type*. The constructor [] denote the usual type constructor of tuple. Two different set constructors $\forall\{\}$, $\exists\{\}$ are introduced. $\forall\{S\}$ corresponds to the usual set constructor and denotes sets whose elements are *all* of type S; $\exists\{S\}$ denotes sets where *at least* one element is of type S. The construct $\sqcap$ holds for

[1] This type must be part of $\mathbf{B}$ because the conjunction of different mono-valued types is empty.

$$
\sigma : \begin{cases}
\sigma(\texttt{Employee}) & = & \triangle[\texttt{name}\colon\texttt{String}, \texttt{salary}\colon 30 \div \infty, \texttt{level}\colon 1 \div 15] \\
\sigma(\texttt{Manager}) & = & \texttt{Employee} \sqcap \triangle[\texttt{salary}\colon 40 \div \infty, \texttt{level}\colon 5 \div 15] \\
\sigma(\texttt{Department}) & = & \triangle[\texttt{name}\colon\texttt{String}, \texttt{managed-by}\colon\texttt{Manager}] \\
\sigma(\texttt{TManager}) & = & \texttt{Manager} \sqcap \triangle[\texttt{level}\colon 8 \div 12] \\
\sigma(\texttt{Material}) & = & \triangle[\texttt{name}\colon\texttt{String}, \texttt{risk}\colon\texttt{Int}, \texttt{feature}\colon\forall\{\texttt{String}\}] \\
\sigma(\texttt{SMaterial}) & = & \texttt{Material} \\
\sigma(\texttt{Storage}) & = & \texttt{Department} \sqcap \triangle[\texttt{stock}\colon\forall\{[\texttt{item}\colon\texttt{Material}, \texttt{qty}\colon 10 \div 300]\}, \texttt{category}\colon\texttt{String}] \\
\sigma(\texttt{SStorage}) & = & \texttt{Storage} \\
\sigma(\texttt{PStorage}) & = & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}\colon(\triangle.\texttt{feature}.\exists\colon \textit{"fluid"}) \sqcap (\triangle.\texttt{risk}\colon 12 \div \infty))
\end{cases}
$$

$$
\mathbf{R} : \begin{cases}
R_1 & = & \texttt{Manager} \sqcap (\triangle.\texttt{level}\colon 5 \div 10) \rightarrow (\triangle.\texttt{salary}\colon 40 \div 60) \\
R_2 & = & \texttt{Material} \sqcap (\triangle.\texttt{risk}\colon 10 \div \infty) \rightarrow \texttt{SMaterial} \\
R_3 & = & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\ \triangle\ .\texttt{feature}.\exists\colon \textit{"F1"}) \rightarrow \triangle[\texttt{managed-by}\colon\texttt{TManager}] \\
R_4 & = & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}\colon\texttt{SMaterial}) \rightarrow \texttt{SStorage} \\
R_5 & = & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{qty}\colon 10 \div 50) \rightarrow (\triangle.\texttt{category}\colon \textit{"A2"})
\end{cases}
$$

Table 1: The Company domain schema

conjunction, whereas $\triangle$, roughly speaking, is an object forming constructor. The type $(p\colon S)$ is called *path type* and represents a short notation of a type always equivalent to a type obtainable with the other type constructors.

Given a set of type descriptions $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$[2], a *schema* σ over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$ is a total function $\sigma : \mathbf{C} \longrightarrow \mathbf{S}$, which associates class names to their descriptions ($\sigma(C) = S$).

We say that C *inherits directly from* C', written $C \prec\!\!\mid C'$, iff $\sigma(C) = C' \sqcap S$ and we require that the inheritance relation is cycle–free, i.e., the transitive closure of $\prec\!\!\mid$, written $\prec$, is a strict partial order. Thus, in $\sigma(C)$ we express both the properties of the class C and the inheritance relation (by means of the $\sqcap$ operator).

Let $\mathbf{R}$ be a countable set of *rules* (denoted by $R_1 R_2 \ldots$). A *rule* R is expressed in the form $R = S^a \rightarrow S^c$ where S^c and S^a represent respectively the *consequent* and the *antecedent* of the rule and are type expressions of $\mathbf{S}$. A rule R is universally quantified over all values $\mathcal{V}(\mathcal{O})$: for all objects v, if v is of type S^a then v must be of type S^c. The Company domain of Section 1, for instance, would lead to the schema specified in Table 1.

2.2 Database Legal Instance, Paths & Rules Semantics

For a given object assignment δ, each type expression S is given a set of values as its interpretation. The *interpretation function* $\mathcal{I}$ is a function from $\mathbf{S}$ to $2^{\mathcal{V}(\mathcal{O})}$ such that: $\mathcal{I}[\top] = \mathcal{V}(\mathcal{O})$, $\mathcal{I}[B] = \mathcal{I}_{\mathbf{B}}[B]$, $\mathcal{I}[C] \subseteq \mathcal{O}$. The interpretation of types is defined inductively for all $S, S' \in \mathbf{S}$ as follows

$$
\begin{aligned}
\mathcal{I}[[a_1\colon S_1, \ldots, a_p\colon S_p]] &= \left\{[a_1\colon v_1, \ldots, a_q\colon v_q] \middle| p \le q, v_i \in \mathcal{I}[S_i], 1 \le i \le p, v_j \in \mathcal{V}(\mathcal{O}), p+1 \le j \le q\right\} \\
\mathcal{I}[\forall\{S\}] &= \left\{\{v_1, \ldots, v_p\} \middle| v_i \in \mathcal{I}[S], 1 \le i \le p\right\} \\
\mathcal{I}[\exists\{S\}] &= \left\{\{v_1, \ldots, v_p\} \middle| \exists i, 1 \le i \le p, v_i \in \mathcal{I}[S]\right\} \\
\mathcal{I}[\triangle S] &= \left\{o \in \mathcal{O} \middle| \delta(o) \in \mathcal{I}[S]\right\} \\
\mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S']
\end{aligned}
$$

[2]In the following, we also will write $\mathbf{S}$ instead of $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$ when the components are obvious from the context.

For the path types we have $\mathcal{I}[(p\colon S)] = \mathcal{I}[(e\colon(p'\colon S))]$ if $p = e.p'$ where

$$\mathcal{I}[(\epsilon\colon S)]=\mathcal{I}[S],\ \mathcal{I}[(a\colon S)]=\mathcal{I}[[a\colon S]],\ \mathcal{I}[(\triangle\colon S)]=\mathcal{I}[\triangle S],\ \mathcal{I}[(\forall\colon S)]=\mathcal{I}[[\forall\{S\}],\ \mathcal{I}[(\exists\colon S)]=\mathcal{I}[\exists\{S\}]$$

The interpretation function does not satisfy the property that the objects of a class be *legal*, i.e. satisfy class descriptions; furthermore it does not satisfy the inclusion relation between type interpretations stated by rules. For this purpose, we have to further constrain the interpretation function as follows. We say that an interpretation function $\mathcal{I}$ is a *legal instance* of a schema σ with rules $\mathbf{R}$ iff the set $\mathcal{O}$ is *finite* and :

$$\mathcal{I}[C] \subseteq \mathcal{I}[\sigma(C)] \quad \text{for all } C \in \mathbf{C} \quad \text{and} \quad \mathcal{I}[S^a] \subseteq \mathcal{I}[S^c] \quad \text{for all } R \in \mathbf{R}$$

i.e., the interpretation of each class *is included* in the interpretation of its description and, for each rule, the antecedent *is included* in the interpretation of the consequent.

Let us give some comments on the introduced semantics. First of all, the interpretation of tuples implies an open world semantics for tuple types, similar to the one adopted by Cardelli [9]. A path type is a type and thus each domain element which satisfies its description belongs to its interpretation; for example, the path type $(\triangle.\texttt{name}\colon \text{"}Martin\text{"})$ individuates all the objects having a property `name` with a *"Martin"* value; these objects might be either employees or departments or materials, since all these classes have an attribute `name`. To restrict to employees, the path type must be conjuncted with the related class name: $\texttt{Employee} \sqcap (\triangle.\texttt{name}\colon \text{"}Martin\text{"})$. Analogously, the path type $S = (\triangle.\texttt{managed-by}.\triangle.\texttt{salary}\colon 40 \div 60)$ do not impose restrictions on the attribute `managed-by`; if we consider this type intersected with the class `Storage`, i.e. $\texttt{Storage} \sqcap S$, $\sigma(\texttt{Storage})$ implicitely imposes that the objects of the domain of `managed-by` belong to the class `Manager`. Furthermore, we can explicitely impose a class as an attribute domain as follows: $S' = (\triangle.\texttt{managed-by}\colon\texttt{TManager} \sqcap (\triangle.\texttt{salary}\colon 40 \div 60))$. Several quantifiers can appear in the same path type. As an example, the path type $S = (\triangle.\texttt{stock}.\forall.\texttt{item}.\triangle.\texttt{feature}.\exists\colon \text{"}solid\text{"})$ individuates all the domain objects such that *all* stocked materials have a feature *"solid"*. On the other hand, the path type $S' = (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{feature}.\exists\colon \text{"}solid\text{"})$ individuates all the domain objects such that *at least* one of the stocked materials has a feature *"solid"*.

2.3 Subsumption in a Database Schema with Rules

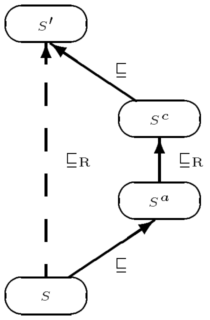
Let us introduce the *subsumption relation* in a schema σ with a set of rules $\mathbf{R}$, written $S \sqsubseteq_{\mathrm{R}} S'$ for $S, S' \in \mathbf{S}$:

$$S \sqsubseteq_{\mathrm{R}} S' \quad \text{iff} \quad \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ for all legal instances } \mathcal{I} \text{ of } \sigma \text{ with rules } \mathbf{R}.$$

It follows immediately that $\sqsubseteq_{\mathrm{R}}$ is a preorder (i.e., transitive and reflexive but antisymmetric) that induces an *equivalence relation* $\simeq_{\mathrm{R}}$ on types: $S \simeq_{\mathrm{R}} S'$ iff $S \sqsubseteq_{\mathrm{R}} S'$ and $S' \sqsubseteq_{\mathrm{R}} S$. Moreover, we say that a type S is *incoherent* iff $S \simeq_{\mathrm{R}} \bot$. This is equivalent to saying that for all domains the type extension is always empty.

Disregarding rules, or at the presence of a schema σ without rules ($\mathbf{R} = \emptyset$), the subsumption relation[3] will be denoted with $\sqsubseteq$. In this paper we extend the subsumption algorithm introduced in [4]. Our aim is to perform $\sqsubseteq_{\mathrm{R}}$ computation by extending $\sqsubseteq$ computation. For this purpose, let us analyze the relationship existing between $\sqsubseteq$ and $\sqsubseteq_{\mathrm{R}}$.

[3]Note that $\sqsubseteq$ is similar to the type *refinement* or *subtyping* relation defined in [9] and adopted for complex object data models in [19] and can be evaluated by syntactical type comparisons.



$$\begin{aligned} S &= \texttt{Manager} \sqcap (\triangle.\texttt{level}\colon 5 \div 8) \\ S' &= (\triangle.\texttt{salary}\colon 30 \div 70) \\ R_1 &= \texttt{Manager} \sqcap (\triangle.\texttt{level}\colon 5 \div 10) \\ &\quad\to (\triangle.\texttt{salary}\colon 40 \div 60) \end{aligned}$$

$$S \sqsubseteq_{\mathrm{R}} S', \quad S \not\sqsubseteq S'$$

Figure 2: Subsumption relations due to the rules.

PROPOSITION 1 *For all* $S, S' \in \mathbf{S}$, *if* $S \sqsubseteq S'$ *then* $S \sqsubseteq_{\mathrm{R}} S'$.

However, the converse, in general, does not hold, as rules state inclusions between type interpretations giving raise to *new* subsumption relations. Intuitively, as shown by the example of figure 2, if $S \sqsubseteq S^a$ and $S^c \sqsubseteq S'$ then $S \sqsubseteq_{\mathrm{R}} S'$. More precisely, given a type S, a rule R and a path p we say that:

> R is *applicable* to S with path p if $S \sqsubseteq (p : S^a)$.

In the example of figure 2, rule R_1 is applicable to S with path ϵ. The same rule is applicable with the path $\triangle.\texttt{managed-by}$ to the type $\texttt{Storage} \sqcap \triangle[\texttt{managed-by}\colon \texttt{Manager} \sqcap (\triangle\texttt{level}\colon 8 \div 10)]$. For rules semantics we have:

PROPOSITION 2 *For all* $S \in \mathbf{S}$ *and for all* $R \in \mathbf{R}$, *if* R *is applicable to* S *with path* p, *we have* $S \sqcap (p\colon S^c) \simeq_{\mathrm{R}} S$.

This transformation is the basic tool for the computation of the $\sqsubseteq_{\mathrm{R}}$ relation. Moreover, the application of a rule may lead to the applicability of other rules. For example, rule R_2 is applicable to $\sigma(\texttt{PStorage})$ with path $\triangle.\texttt{stock}.\exists.\texttt{item}$, obtaining $\sigma(\texttt{PStorage}) \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}\colon \texttt{SMaterial})$; to this type is applicable rule R_4 with path ϵ, obtaining $\sigma(\texttt{PStorage}) \sqcap \texttt{SStorage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}\colon \texttt{SMaterial})$. As a result, $\texttt{PStorage} \sqsubseteq_{\mathrm{R}} \texttt{SStorage}$ is computed (see figure 1).

To individuate all the trasnformations of a type implied by the given set of rules $\mathbf{R}$, we define the function $\Gamma : \mathbf{S} \longrightarrow \mathbf{S}$, as follows:

$$\Gamma(S) \quad = \quad \begin{cases} S \sqcap (p : S^c) & \text{if it exists } R \text{ and } p \text{ such that } S \sqsubseteq (p\colon S^a) \text{ and } S \not\sqsubseteq (p\colon S^c) \\ S & \text{otherwise} \end{cases}$$

Given a type S, we define the *semantic expansion* of S, $EXP(S)$, as follows: $EXP(S) = \Gamma^i(S)$, where i is the least integer[4] such that $\Gamma^i(S) = \Gamma^{i+1}(S)$.

PROPOSITION 3 *For all* $S \in \mathbf{S}$ *we have* $EXP(S) \simeq_{\mathrm{R}} S$ *and for all* S' *such that* $S' \simeq_{\mathrm{R}} S$ *we have* $EXP(S) \sqsubseteq S'$.

$EXP(S)$ allows the formalization of the relationship between the $\sqsubseteq_{\mathrm{R}}$ and $\sqsubseteq$ relations:

THEOREM 1 *For all* $S, S' \in \mathbf{S}$ *we have* $S \sqsubseteq_{\mathrm{R}} S'$ *if and only if* $EXP(S) \sqsubseteq S'$.

[4] The existence of i is guaranteed as the number of rules is finite and a rule cannot be applied more than once with the same path ($S \not\sqsubseteq (p\colon S^c)$).

Therefore, computing subsumption in a schema with rules can be performed by firstly determining the semantic expansion of a type and, secondly, by computing subsumption in a schema disregarding rules. A subsumption algorithm, which extends the one proposed in [4], by considering also existentially quantified set constructors, is given in the following.

Computing subsumption between types is a quite difficult task. Instead of directly specifying an algorithm for this problem, a detour will be taken, defining an algorithm on *canonical extensions* of a given schema. Let $\mathbf{N}$ $(N, N', \ldots)$ be a set of names denoting type descriptions, $\breve{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ be a set of *canonical type descriptions* $(\breve{S}, \breve{S}', \ldots)$ and ν be a *canonical schema* over $\breve{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ such that $\nu(N) = \breve{S}$. To compute subsumption, we define *conditional subsumption* based on a relation $\leq \subseteq \mathbf{N} \times \mathbf{N}$. We say that $\breve{S}$ is *conditionally subsumed* by $\breve{S}'$ based on $\leq$, written $\breve{S} \sqsubseteq_{\leq} \breve{S}'$ under the following conditions:

$$\begin{array}{rcl}
B \sqsubseteq_{\leq} B' & \text{iff} & B \sqcap B' = B \\
\forall\{N\} \sqsubseteq_{\leq} \forall\{N'\} & \text{iff} & N \leq N' \\
\bigsqcap_i \exists\{N_i\} \sqcap \forall\{N\} \sqsubseteq_{\leq} \forall\{N'\} & \text{iff} & N \leq N' \\
\bigsqcap_i \exists\{N_i\} \sqcap \forall\{N\} \sqsubseteq_{\leq} \bigsqcap_j \exists\{N_j'\} \sqcap \forall\{N'\} & \text{iff} & N \leq N' \wedge \forall j \exists i{:}\, N_i \leq N_j' \\
[\ldots, a_i{:}\, N_i, \ldots] \sqsubseteq_{\leq} [\ldots, a_j'{:}\, N_j', \ldots] & \text{iff} & \forall j \exists i{:}\ (a_i = a_j' \wedge N_i \leq N_j') \\
\bigsqcap_i \overline{C}_i \sqcap \triangle N \sqsubseteq_{\leq} \bigsqcap_j \overline{C}_j' \sqcap \triangle N' & \text{iff} & N \leq N' \wedge \forall j \exists i{:}\, \overline{C}_i = \overline{C}_j'.
\end{array}$$

Algorithm 1 (Subsumption) *Let $\leq^0 = \mathbf{N} \times \mathbf{N}$ and define $\leq^{i+1}$ in the following way:*

$$\leq^{i+1} = \left\{ (N, N') \in \mathbf{N} \times \mathbf{N} \,\middle|\, \nu(N) \sqsubseteq_{\leq^i} \nu(N') \vee N = \bot \vee N' = \top \right\}.$$

Compute $\leq^0, \ldots, \leq^i, \ldots$ until $\leq^i = \leq^{i+1}$ and set $\lesssim_\nu = \leq^i$.

A natural number n always exists such that $\leq^n = \leq^{n+1}$ and the maximal n is $|\mathbf{N}|^2$, i.e., the subsumption algorithm is polynomial in the size of the canonical schema ν. The following theorem states correctness and completness of the subsumption algorithm.

Theorem 2 *Given a canonical schema ν over $\breve{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, for all $N, N' \in \mathbf{N}$: $N \sqsubseteq N'$ iff $N \lesssim_\nu N'$.*

As any schema σ can be effectively transformed into an *equivalent* canonical schema ν (see Appendix), it follows that subsumption in an arbitrary schema can be computed by computing $\lesssim_\nu$ on its canonical form.

As a last point, let us comment the relationship existing between the $\sqsubseteq_{\mathrm{R}}$ relation and the inheritance relation: 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': consequently, if $C \prec C'$ then $C \sqsubseteq_{\mathrm{R}} C'$. The converse, in general, does not hold, the main reason being the presence of rules. For example, if we consider the classes `PStorage`, we have `PStorage` $\sqsubseteq_{\mathrm{R}}$ `SStorage` (by applying rules R_2 and R_4) and `PStorage` $\not\prec$ `SStorage`.

3 Semantic Query Optimization

(The clean part of) a query is an object–type S with the semantics of a virtual class, usually expressed as the conjunction of one or more *target* class names (that is the classes from which we want to select objects) and one or more path types (which further restricts the set of objects to be selected from the classes). Thus, a query is expressible as a conjunction of path types factors $\sqcap_i^n (p_i : S_i)$ and its result is exactly the set of the domain objects which belongs to the S extension, i.e. $\mathcal{I}[S]$.

In our theoretical framework, given a query Q, we perform semantic expansion by means of the EXP function, efficiently computing implication by the subsumption algorithm.

PROPOSITION 4 *Given a query Q and a path type factor $(p: S)$, $(p: S)$ can be added to Q, preserving semantic equivalence ($Q \sqcap (p: S) \simeq_{\mathrm{R}} Q$) iff $Q \sqsubseteq_{\mathrm{R}} (p: S)$.*

The example of Section 1 (query Q) showed that we can obtain an optimized query by *specializing* the classes involved in a query. Le us show how it is obtained by the exploiting the canonical form and the EXP function. Firstly, we obtain the canonical form of the query Q ($\nu(Q)$), as follows:

$$
\begin{aligned}
Q &= \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk}: 15 \div \infty) \\
\nu(Q) &= (\overline{\texttt{Department}} \sqcap)\ \overline{\texttt{Storage}} \sqcap \triangle N_1 \\
\nu(N_1) &= [\texttt{name}: \texttt{String}, \texttt{managed-by}: N_2, \texttt{stock}: \forall\{N_3\} \sqcap \exists\{N_4\}, \texttt{category}: \texttt{String}] \\
\nu(N_2) &= (\overline{\texttt{Employee}} \sqcap)\ \overline{\texttt{Manager}} \sqcap \triangle[\texttt{name}: \texttt{String}, \texttt{salary}: 40 \div \infty, \texttt{level}: 5 \div 15] \\
\nu(N_3) &= [\texttt{item}: N_5, \texttt{qty}: 10 \div 300] \qquad \nu(N_4) = [\texttt{item}: Q_1, \texttt{qty}: 10 \div 300] \\
\nu(N_5) &= \overline{\texttt{Material}} \sqcap \triangle[\texttt{name}: \texttt{String}, \texttt{risk}: \texttt{Int}, \texttt{feature}: \forall\{\texttt{String}\}] \\
\nu(Q_1) &= \overline{\texttt{Material}} \sqcap \triangle[\texttt{name}: \texttt{String}, \texttt{risk}: 15 \div \infty, \texttt{feature}: \forall\{\texttt{String}\}]
\end{aligned}
$$

Note that conjunctions between base types (see $\texttt{salary}: 30 \div \infty$, $\texttt{salary}: 40 \div \infty$) have been solved; the path type ($\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk}: 15 \div \infty$) has been subdivided to deal with the different semantics of $\exists$ and $\forall$ quantifiers. Furthermore, we solved conjunctions between imaginary atomic classes on the basis of the $\sqsubseteq_{\mathrm{R}}$ relationship: $\overline{C} \sqcap \overline{C}' = \overline{C}$ if $C \sqsubseteq_{\mathrm{R}} C'$.[5] Thus, by generating the canonical form of a query we perform the *semantic transformation* due to the inheritance hierarchy as in [24]. For instance, $\overline{\texttt{Storage}} \sqcap \overline{\texttt{Department}} = \overline{\texttt{Storage}}$ as $\texttt{Storage} \sqsubseteq_{\mathrm{R}} \texttt{Department}$. On $\nu(Q)$ we apply the EXP function (rules R_2 and R_4), obtaining:

$$
\begin{aligned}
\nu(Q_1) &= \overline{\texttt{SMaterial}} \sqcap \triangle[\texttt{name}: \texttt{String}, \texttt{risk}: 15 \div \infty, \texttt{feature}: \forall\{\texttt{String}\}] \\
\nu(Q) &= \overline{\texttt{SStorage}} \sqcap \triangle N_1
\end{aligned}
$$

Note that `SStorage` (`SMaterial`) are the *most specialized generalization* classes for Q (Q_1). More generally, given the canonical form of a query $\nu(Q) = \sqcap_i C_i \sqcap \triangle N$, N may include sub–queries, say $N_k = \sqcap_j C_{k,j} \sqcap \triangle N_k'$. Given $(p_k: N_k)$, $k \geq 0$, (for $k = 0$, $p_0 = \epsilon$ e $N_0 = \nu(Q)$) the set of the *most specialized generalization* classes for a query and its sub–queries, is defined as follows:

$$
MSG_Q(N_k) = \{C \mid Q \sqsubseteq_{\mathrm{R}} (p_k: C), \nexists C', C' \neq C, Q \sqsubseteq_{\mathrm{R}} (p_k: C'), C' \sqsubseteq_{\mathrm{R}} C\}
$$

$MSG_Q(N_k)$ is directly given by $\nu(EXP(Q))$.

The following example shows another possibility of semantic expansion which may be useful for query optimization, the so called *index introduction*: i.e the introduction of a factor including an indexed attribute. Q_2: Select storages storing *at least* a quantity $(10 \div 40)$ of the material named *"iron"*:

[5] the terms enclosed in brackets are shown only for clarity since they are discarded on the basis of the $\sqsubseteq_{\mathrm{R}}$ relationship.

$Q_2 = \texttt{Storage} \sqcap\ \triangle[\texttt{stock}{:}\,\exists\{[\texttt{item}{:}\,\texttt{Material} \sqcap (\triangle.\texttt{name}{:}\ \textit{"iron"}), \texttt{qty}{:}\,10 \div 40]\}$

Rule R_3 is applicable to Q_2 with ϵ obtaining $EXP(Q_2) = Q_2 \sqcap (\triangle.\texttt{category}{:}\ \textit{"A2"})$. If informations about indexes on schema attributes are available and `category` is an indexed attribute, the introduced path type can be used in processing the query.

Another rewriting rule proposed in [21, 23] is the *constraint removal*: if the query minus some factors f_i implies the antecedent of a rule and the consequent of that rule implies the f_i factors then the f_i factors can be removed from the query. In our approach we can formalize *constraint removal* as follows:

Proposition 5 *Given a conjunction of path types* $\sqcap_i^n (p_i{:}\, S_i)$ *we have that* $\sqcap_{j\neq i}^{n-1} (p_j{:}\, S_j) \simeq_{\mathrm{R}} \sqcap_i^n (p_i{:}\, S_i)$ *iff* $\sqcap_{j\neq i}^{n-1} (p_j{:}\, S_j) \sqsubseteq_{\mathrm{R}} (p_i{:}\, S_i)$.

As an example, let us introduce the query Q_4:

Q_3: Select managers with a level from 5 to 8 and earning a salary from 30K to 70K:

$Q_3 = \underbrace{\texttt{Manager} \sqcap (\triangle.\texttt{level}{:}\,5 \div 8)}_{S} \sqcap \underbrace{(\triangle.\texttt{salary}{:}\,30 \div 70)}_{S'}$

As shown in figure 2, we have that $S \sqsubseteq_{\mathrm{R}} S'$; consequently the factor S' can be eliminated obtaining: $Q_3^{'} = \texttt{Manager} \sqcap (\triangle.\texttt{level}{:}\,5 \div 8)$. If a factor determinates the navigation through a number of classes, its elimination avoids the access to those classes. As an example, consider the following query:

Q_4: Select storages managed by managers with a level$(6 \div 14)$ and storing at least a material with a feature F1:

$Q_4 = \underbrace{\texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{feature}.\exists{:}\ \textit{"F1"})}_{S} \sqcap \underbrace{(\triangle.\texttt{managed-by}.\triangle.\texttt{level}{:}\,6 \div 14)}_{S'}$

By rule R_3 and by the description of `TManager`, we have that $S \sqsubseteq_{\mathrm{R}} S'$,therefore the factor S' can be eliminated from the query avoiding the access to the class `Manager`.

References

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

[2] J.P. Ballerini and S. Bergamaschi. Automatic building and validation of complex object database schemata. Technical Report 5/124, CNR - Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, July 1993.

[3] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. Technical Report 103, CIOC–CNR, Viale Risorgimento, 2 Bologna, Italia, September 1994.

[4] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[5] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.

[6] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Object-oriented query languages: The notion and the issues. *IEEE Trans. Knowl. and Data Engineering*, 4(3):223–236, June 1992.

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

[8] M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.

[9] L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.

[10] 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.

[11] Edward P.F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Principles of Database Systems*, pages 202–11. ACM, 1992.

[12] 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.

[13] M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.

[14] M. Jeusfeld and M. Staudt. Query optimization in deductive object bases. In Freytag, Maier, and Vossen, editors, *Query Processing for Advanced Database System*. Morgan Kaufmann Publishers, Inc., June 1993.

[15] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *SIGMOD '92*, pages 393–402. ACM, June 1992.

[16] W. Kim. A model of queries for object-oriented database systems. In *Int. Conf. on Very Large Databases*, Amsterdam, Holland, August 1989.

[17] J. J. King. *Query optimization by semantic reasoning*. PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.

[18] J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.

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

[20] H. H. Pang, H. Lu, and B.C. Ooi. An efficient semantic query optimization algorithm. In *Int. Conf. on Data Engineering*, pages 326–335, 1991.

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

[22] S. Shenoy and M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. and Data Engineering*, 1(3):344–361, September 1989.

[23] 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.

[24] D. D. Straube and T. Özsu. Queries and query processing in object-oriented database systems. *ACM Trans. on Information Systems*, 8(4):387–430, October 1990.

Appendix

Canonical Form

Let $\overline{\mathbf{C}}$ ($\overline{C}, \overline{C}', \ldots$) be the set of imaginary atomic classes, which are a disjoint copy of $\mathbf{C}$ and $\mathbf{N}$ be the set of names denoting type descriptions which contains $\overline{\mathbf{C}}$. We define the set $\check{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ of *canonical type descriptions* $\check{S}$ as follows ($N, N_k, N_j \in \mathbf{N}, \overline{C}_i \in \overline{\mathbf{C}}, k \geq 0, j \geq 1, i \geq 0$):

$$\check{S} \quad \rightarrow \quad \top \mid B \mid N \mid [a_1 : N_1, \ldots, a_k : N_k] \mid \forall\{N\} \mid \sqcap\exists\{N_j\} \sqcap \forall\{N\} \mid \sqcap\overline{C}_i \sqcap \triangle N$$

A *canonical schema* ν is a schema over $\check{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ ($\nu(N) = \check{S}$). An interpretation $\mathcal{I}'$ of the set $\check{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ is straightforward defined by the interpretation formulas of section 2.2 for types, but substituting class names interpretation with a more general names interpretation formula: $\mathcal{I}[N] \subseteq \mathcal{V}(\mathcal{O})$. We say that an interpretation $\mathcal{I}'$ is a legal instance of the schema ν iff, for all $N \in \mathbf{N}$, we have $\mathcal{I}'[N] = \mathcal{I}'[\nu(N)]$.

Given a schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ and a schema ν over $\check{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, let $\mu : \mathbf{S} \rightarrow \mathbf{N}$ be a function which associates to each type $S \in \mathbf{S}$ a name N. We say that μ is a *canonical transformation* di σ iff for any legal instance $\mathcal{I}$ of σ a legal instance $\mathcal{I}'$ of ν exists and *vice versa* such that: $\mathcal{I}[S] = \mathcal{I}'[\mu(S)]$ for all $S \in \mathbf{S}$. Note that this implies that the subsumption relations in σ ($\sqsubseteq_\sigma$) and in σ ($\sqsubseteq_\nu$) are identical: $S \sqsubseteq_\sigma S'$ iff $\mu(S) \sqsubseteq_\nu \mu(S')$.

PROPOSITION 6 *For any schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ can be effectively define a schema ν over $\check{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ and a function μ that is a* canonical transformation *of σ.*

The algorithm for transforming a given schema into an equivalent canonical schema is given in [2]. The main points are: every class C is expressed as the conjunction of an imaginary atomic class $\overline{C}$, which belongs to the set $\overline{\mathbf{C}}$, and its description $\sigma(C)$; a path type $(p{:}\, S)$ is unfolded by means of the other type constructors; the description of every class is transformed by substituting only the names of classes from which it inherits with their descriptions and by applying suitable conjunction rules together with the denomination of new types obtained as conjunctions. Therefore, at first $\mathbf{N} = \mathbf{C}$, but this set of names grows with the transformations made.

Schema Coherence

The problem of avoiding contradictions and redundancies in a set of integrity rules and, more generally, in a schema including rules, represent an important component of any semantic query optimization system, as pointed out in [22]. We investigated this problem in [3], proposing this solution: rules are embedded in type descriptions (by means of the union and the complementation operators); a specialized reasoner, based on the *tableaux calculus*, able to compute subsumption and check coherence for the type system, is provided.

From theorem 1, it follows that coherence of a type S can be checked by means of $EXP(S)$:

COROLLARY 1 *For all $S \in \mathbf{S}$, S is incoherent ($S \simeq_{\mathrm{R}} \perp$) iff $EXP(S) \simeq \perp$, where $\simeq$ is the equivalence relation in a schema disregarding rules.*

A schema σ with a set of rules $\mathbf{R}$ is called *coherent* if the set $\mathbf{R}$ of rules is *non–contradictory* and *non–redundant* and it does not contain any incoherent class name.