Subsumption between Queries to Object-Oriented Databases*

Martin Buchheit[†] Manfred A. Jeusfeld[‡] Werner Nutt[†]
Martin Staudt[‡]

[†]German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, {buchheit,nutt}@dfki.uni-sb.de
[‡]RWTH Aachen, Informatik V, Ahornstr. 55, D-52056 Aachen, Germany, {jeusfeld,staudt}@informatik.rwth-aachen.de

Abstract

Most work on query optimization in relational and object-oriented databases has concentrated on tuning algebraic expressions and the physical access to the database contents. The attention to semantic query optimization, however, has been restricted due to its inherent complexity. We take a second look at the problem for queries in object-oriented databases and find that reasoning techniques for concept languages developed in Artificial Intelligence apply for the following reasons: concept languages have been tailored for efficiency and their semantics is compatible with class and query definitions in object-oriented databases. We propose a query optimizer which decides subset relationships between a query and a view (a simpler query whose answer is stored) in polynomial time.

*This work was supported in part by the Commission of the European Communities under ESPRIT Basic Research Action 6810 (Compulog 2), by the German Ministry of Research and Technology under grant ITW 92-01 (TACOS), and by the Ministry of Science and Research of Nordrhein-Westfalen.

1 Introduction

From an abstract viewpoint, databases organize information in sets of objects. In object-oriented databases (OODB's), these sets are called *classes* and the elements are constrained by some—not too complicated—type expression.

Similar expressions for describing classes of objects—so-called *concept descriptions* or simply *concepts*—have been investigated in Artificial Intelligence (AI), where they occur in knowledge representation languages of the KL-ONE family [WS92]. This research has come up with techniques to determine consistency and subsumption of concepts and has assessed the complexity of these inferences for a variety of languages (see *e.g.*, [DLNN91a]).

We believe that this relationship between work on databases and AI offers a potential of cross-fertilization: database research can profit from reasoning techniques for concepts, and knowledge representation research can learn about properties of practically applied set descriptions like class schemas and queries in OODB's.

The Problem. Query optimization is largely reasoning on intentional representations, esp. queries and schema information. Therefore, this area appears as a natural choice for proving the hypothesis of cross-fertilization. The key problem we want to solve in this paper is the following:

> *Given the schema of an OODB and two queries, decide whether in every possible state the answer set of the first query is contained in the answer set of the second.*

Subsequently, the second query will be called a *view*. Answers to views are precomputed and maintained to be up-to-date. If the answer to a view forms a superset of those objects retrieved by a query this fact may be used to speed up query evaluation by filtering the results of the view, instead of computing the answers from scratch.

OODB's offer increased opportunities to reuse queries: their schema is usually richer, *i.e.*, more detailed, than the schema of relational databases. In particular, attribute values are constrained by types (or classes). Classes may have subclasses with additional properties and/or constraints. We argue that this information should be utilized for query optimization.

In order to exploit subsumption of queries by views in a real system, many views might have to be checked for a given query, and checks will be performed for every incoming query. Hence, efficiency is an issue, and subsumption should be decidable in polynomial time.

The Schema and Query Language. Regrettably, there exists neither a standardized object-oriented data model nor a standardized object-oriented query language. We therefore present a language that features just the properties which are common to most object-oriented data models *and* are relevant to our purpose.

- *Membership of objects in classes:* Classes group a finite set of objects (their instances). In most systems, membership of an object in a class is constrained by the type of the class. Here, we assume that the condition for membership can be expressed in first order logic.
- *Subclass relationships:* Classes are organized in a subclass hierarchy. Any instance of a class is also an instance of the superclasses.
- *Attribute definitions:* Objects may have attributes. Domain and range of an attribute are restricted by classes. For a subclass of its domain, an attribute may be restricted to take values in a subclass of its range.
- *Number restrictions:* The attribute values can be singletons (at most one value), sets (finetely many values), and necessary (at least one value). The latter one is very important for OODB's because it prevents method executions from errors when accessing such attributes.

Our Approach to Solving the Problem. The complexity of query and schema languages in OODB's makes a general solution of our problem impossible. Thus, it has to be relaxed to a simpler problem. We identify portions of the schema and queries which can be mapped to a concept language where subsumption can be decided efficiently. Concept languages[1] are fragments of first-order logic: concepts can be viewed as certain logical formulas that are built using unary and binary predicates and contain one free variable (to be filled with instances of the concept). They bear strong similarities to languages for describing schemas and queries of OODB's. Subsumption of concepts has been studied for a variety of languages and the borderline between variants where reasoning is tractable and where not is by now well understood (see *e.g.*, [DLNN91b]). Three points make the ideas of our approach more precise:

- *Clean and dirty parts of queries:* A query is separated into a *clean* part that can be represented with the concept language, and a *dirty* part that goes beyond concept language expressibility. Structurally object-oriented properties like attribute typing and path expressions fall into this clean part.
- *Views as clean queries:* In this paper, views are queries whose answers are precomputed. The answers to a view can be used for optimizing queries subsumed by the view. In order to keep our approach sound, we forbid dirty parts for views.
- *Incorporation of schema information:* The schema of the database is explicitly employed for checking subsumption between two queries. Object-oriented schemas are comparatively large and mainly built using constructs which still permit polynomial time algorithms for subsumption.

As a consequence, completeness is sacrificed for efficiency: not all subsumptions valid in the more general languages are detected. However, since the amount of information lost by neglecting the dirty part is rather small it can be expected that the "hit rate" is still high enough for justifying the effort.

The following section defines a generic syntax and semantics for object-oriented schemas and queries. Section 3 introduces a concept language for the clean parts of schemas and queries. The subsumption calculus together with a proof for completeness and soundness is elaborated in Section 4. Section 5 discusses related work.

2 Object-Oriented Databases and Queries

In this section, we introduce a simple frame-like database language $\mathcal{DL}$ which on the one hand provides a generic data model for OODB's and on the other hand has a simple first-order semantics. The language incorporates the three basic abstraction principles *classification*, *generalization*, and *aggregation*. Subsection 2.1 describes the language used for defining the database schema. In Subsection 2.2 we show how queries can naturally be represented in this framework as special classes.

2.1 Defining an OODB Schema

A $\mathcal{DL}$ schema consists of a set of attribute and class declarations. For an example, Figure 1 demonstrates part of the schema of a medical database containing patients who are persons suffering from diseases, taking drugs, and consulting doctors.[2]

[1]Concept languages are also known as *Terminological Logics* or *Description Logics.*

[2]A complete schema must contain a declaration for every class and attribute. A completion of our example would therefore contain additional declarations for the classes Drug, Disease, String, and Topic, as well as for the attributes consults, name, suffers, and takes.

```
Class Patient isA Person with
  attribute
    takes: Drug
    consults: Doctor
  attribute, necessary
    suffers: Disease
  constraint:
    forall p/Patient
    not (p in Doctor)
end Patient

Class Person with
  attribute, necessary, single
    name: String
end Person

Class Doctor with
  attribute
    skilled_in: Disease
end Doctor

Attribute skilled_in with
  domain: Person
  range: Topic
  inverse: specialist
end skilled_in
```

Figure 1: A part of the schema of a medical database

Attributes are typed binary relations with specific classes as domains and ranges. It is allowed to define synonyms for the inverse of an attribute. Thus skilled_in is a relation between persons and general topics whose inverse is called specialist. Synonyms for attributes are not allowed to occur in other declarations of a schema, but are useful for formulating queries. *Classes* group objects, which are restricted by specific conditions. These conditions are necessary, but not sufficient, *i.e.*, an object is not automatically recognized as a member if it satisfies the restrictions of a class.

First of all, classes may be specializations, respectively *generalizations*, of other classes. This is indicated by isA-statements in class declarations. In our example, Patient is a subclass of Person. As a consequence, in any legal state of the database, every patient must be a person. There is a most general class OBJECT containing any object of the database.

Second, class declarations state typical properties of their members (*aggregation*), which are expressed through attributes. For members of the class declared, the values of an attribute may be restricted to a subclass of the attribute's range. Thus, in any legal state of our example database, only members of class Disease are admissible as values for the skilled_in attribute of a Doctor. Besides this typing condition, attributes of a class may be declared as mandatory (necessary) or functional (single). In our example, a patient always must suffer from at least one disease, while a person must have exactly one name.

Finally, general integrity constraints that class members have to obey can be stated in a constraint clause. Constraints are formulated in a first-order many sorted language where quantifiers are restricted to range over classes. The only atoms allowed in this language are (x in C), denoting membership of x in class C, and (x a y), assigning y as value for attribute a to object x. The example constraint on the class Patient forbids a person to be both a patient and a doctor. In the following, we will refer to the subclass and attribute part of a class declaration as the *structural* or *clean* and to the constraint part as the *dirty* part. The latter is not taken into account by the abstraction in the Section 3.

We will not go into the details of how to specify a *state* of a $\mathcal{DL}$ database. This can be done *e.g.*, using similar frame-like constructs relating objects to classes by instance-relationships (*classification*) and to each other by assigning values to attributes defined for these classes.

The semantics of the language is given by a mapping from attribute and class declarations to first-order formulas, where class names appear as unary and attribute names as binary predicates. Facts about database objects are mapped to sets of ground atoms built from these predicates. We do not specify the mapping formally, but translate in Figure 2 as an example the declarations of the class Patient and the attribute skilled_in.

We assume that every state of the database gives rise to exactly one model of these formulas. This might be achieved in several ways: either all facts are explicitly stated, or some schema formulas are employed as deductive rules, by which additional facts are derived (see *e.g.*, [SNJ93]). The important point is that every state defines a unique structure that satisfies the schema.

$$\begin{array}{ll}
\forall x.\,\mathsf{Patient}(x) \Rightarrow \mathsf{Person}(x) & \forall x,y.\,\mathsf{skilled_in}(x,y) \Rightarrow \\
\forall x,y.\,\mathsf{Patient}(x) \wedge \mathsf{takes}(x,y) \Rightarrow \mathsf{Drug}(y) & \qquad\qquad \mathsf{Person}(x) \wedge \mathsf{Topic}(y) \\
\forall x,y.\,\mathsf{Patient}(x) \wedge \mathsf{consults}(x,y) \Rightarrow \mathsf{Doctor}(y) & \forall x,y.\,\mathsf{skilled_in}(x,y) \Leftrightarrow \mathsf{specialist}(y,x) \\
\forall x,y.\,\mathsf{Patient}(x) \wedge \mathsf{suffers}(x,y) \Rightarrow \mathsf{Disease}(y) & \\
\forall x.\,\mathsf{Patient}(x) \Rightarrow \exists y\,\mathsf{suffers}(x,y) & \\
\forall x.\,\mathsf{Patient}(x) \Rightarrow \neg\,\mathsf{Doctor}(x) &
\end{array}$$

Figure 2: Translating declarations into logic

2.2 Query Classes

Querying a database means retrieving stored objects that satisfy certain restrictions or qualifications and hence are interesting for a user. In relational databases, queries are constructed by algebra expressions involving relations from the database, and answers again are relations, *i.e.*, sets of tuples. The correspondence between database and answer format has obvious advantages. In OODB's classes are used to represent sets of objects and thus it is natural to use them also for describing query results.

Object-oriented data models disagree as to whether new objects can be created as answers to queries or not (see *e.g.*, [AK89]). In the object model presented here we restrict answer objects to existing objects[3] that are deduced as instances of so-called *query classes*. In contrast to the classes constituting the database schema the membership conditions in the declarations of query classes are necessary *and* sufficient. Thus, they are completely defined by the declaration, and objects can be recognized as instances although they have not explicitly been entered as such.

An example for a query class is given in Figure 3. Just as schema classes, query classes may be specializations of other classes, especially query classes, and answer objects must be common instances of all superclasses. Hence, every schema class can be turned into a query class.

In order to express more specific conditions on answer objects, so-called *derived* objects can be specified in the derived clause through labeled paths. A *labeled path* is a labeled chain of attributes[4] with value restrictions and has the form

$$\mathsf{l}_j\colon\ (\mathsf{a}_1\colon\ \mathsf{C}_1).(\mathsf{a}_2\colon\ \mathsf{C}_2).\ldots.(\mathsf{a}_n\colon\ \mathsf{C}_n),$$

where l_j is the label, the a_i's are attributes, and each C_i is a class D or a singleton set $\{\mathsf{i}\}$. Labels stand for derived objects. A restricted attribute $(\mathsf{a}\colon\ \mathsf{D})$ ($(\mathsf{a}\colon\ \{\mathsf{i}\})$) relates all objects x, y in the database such that y is an instance of D ($y = \mathsf{i}$) and an a-value of x. If an attribute is restricted by OBJECT then the restriction can be omitted. The chain of attributes in a labeled path can be conceived as a new attribute obtained by composing the components of the chain. Intuitively, for a given object, a chain denotes the set of objects that can be reached following it. The label of a path can be viewed as a variable ranging over this set, and for an object to be an instance of the query class this variable has to be bound to some element of the set. Summarizing, our labeled paths generalize the common notion of paths (see *e.g.*, [KKS92]) in that they allow one after each step to filter the values of an attribute. Moreover, one can access the values at the end of the path through a label.

The where clause contains equalities $\mathsf{l}_j = \mathsf{l}_k$ between labels,[5] which the derived objects have to satisfy.

[3] For simplicity we ignore additional output attributes of answer objects in this paper.

[4] Attribute synonyms defined in the schema are allowed in paths, too.

[5] In order to keep the presentation of our algorithm as simple as possible we will consider only the case that a label occurs no more than once in the where clause.

```
QueryClass QueryPatient isA Male, Patient with                  (1)
  derived
    l_1: (consults: Female)                                     (2)
    l_2: suffers.(specialist: Doctor)                           (3)
  where
    l_1 = l_2                                                   (4)
  constraint:
    forall d/Drug not (this takes d) or (d = Aspirin)           (5)
end QueryPatient
```

Figure 3: A query

Finally, query classes contain a **constraint** clause (again called *dirty part*) where additional conditions for class membership are specified by a logical formula similar to the one in schema class constraints. In the formula the labels l_j may appear again. The variable **this** refers to the answer object itself. Labels that occur neither in the **where** nor the **constraint** clause may be omitted in the **derived** part.

In the example, **QueryPatient** retrieves all patients from the database who consult a female who is a doctor and a specialist in the disease from which the patient is suffering. In addition, the patients do not take any drug except Aspirin.

Again, the semantics of query classes is given by a translation into a predicate logic formula. The formula conjoins the membership predicates for the superclasses, subformulas gained from the labeled paths, equalities, and a straightforward rewriting of the constraint. The query **QueryPatient** yields the formula given in Figure 4. Each conjunct corresponds to the clause with the same number within the definition of **QueryPatient**. In a framework that combines deductive databases and object-orientedness, the translated query class can be readily executed (see [SNJ93]).

Based on the entire conceptual schema of a database, users and application programs usually work on subschemas that constitute their *external view* on the database. A common approach (*e.g.*, in SQL) is to use for the definition of such views a sublanguage of the query language. Thus query classes as introduced above do in fact define simple views on $\mathcal{DL}$ databases with a given schema. If views are used frequently and the computation of their extension is expensive they can be *materialized*. Materialization means that membership of objects in a view, although derivable from the database by means of the view definition (and hence redundant), is explicitly stored. On the one hand, direct access to materialized views is as fast as to any other class defined in the schema. On the other hand, since views are just special queries with stored answers, the detection that a view subsumes a query, allows one to profit from this fast access by restricting the search space for query evaluation just to the stored instances of the view.

$$
\begin{array}{ll}
\mathsf{QueryPatient}(t) \iff \mathsf{Male}(t) \wedge \mathsf{Patient}(t) \wedge & (1)\\
\quad \exists\, l_1, l_2. & \\
\qquad \mathsf{consults}(t, l_1) \wedge \mathsf{Female}(l_1) \wedge & (2)\\
\qquad \exists x.\, (\mathsf{suffers}(t, x) \wedge \mathsf{specialist}(x, l_2) \wedge \mathsf{Doctor}(l_2)) \wedge & (3)\\
\qquad l_1 \doteq l_2 \wedge & (4)\\
\qquad \forall\, d.\, (\mathsf{Drug}(d) \Rightarrow \neg \mathsf{takes}(t, d) \vee d \doteq \mathsf{Aspirin}) & (5)
\end{array}
$$

Figure 4: Translating the query into logic

```
QueryClass ViewPatient isA Patient with
   derived
         (name: String)
     l_1: (consults: Doctor).(skilled_in: Disease)
     l_2: (suffers: Disease)
   where
     l_1 = l_2
end ViewPatient
```

Figure 5: A view

Let us extend our example by a second query class, defined in Figure 5. The class ViewPatient is a view defining another subset of patients in the database, namely those whose name is stored and that consult a doctor who is a specialist for one of their diseases. At first glance it is not obvious whether the ViewPatient subsumes QueryPatient. However, if one takes into account the schema information that (1) every person and hence every patient has a name, that (2) patients suffer from diseases, and (3) the attributes skilled_in and specialist are inverses of each other, moreover, if one joins the paths as required by the labels, one realizes that every instance of QueryPatient is also an instance of ViewPatient.

3 From Queries to Concepts

In this section we introduce two languages for describing schemas and queries, respectively, that are abstractions of $\mathcal{DL}$. While the frame-based syntax of $\mathcal{DL}$ is user-oriented and similar to that of languages in existing OODB systems the syntax of the abstract languages is well suited for the design, verification and complexity analysis of algorithms. It is inspired by KL-ONE-like concept languages and employs a variable free notation which is semantically equivalent to certain logical formulas. The lack of variables keeps these languages close to the clean part of the concrete language while the explicit use of quantifiers resolves ambiguities present in $\mathcal{DL}$. We will show how to represent the clean parts of class and attribute declarations by a set of *schema axioms* and query classes as *concepts* in the abstract languages. Then we reformulate the key problem of the paper for the new languages.

3.1 Preliminaries

Here we specify syntax and semantics of the concept languages $\mathcal{SL}$ and $\mathcal{QL}$. The elementary building blocks are primitive concepts (ranged over by the letter A) and primitive attributes (ranged over by P). Intuitively, concepts describe sets and thus correspond to unary predicates while attributes describe relations and thus correspond to binary predicates. We assume also that an alphabet of *constants* (ranged over by a, b, c) is given.

In the *schema language* $\mathcal{SL}$ attributes must be primitive. Concepts (ranged over by C, D, E) in $\mathcal{SL}$ are formed according to the following syntax rule:

$$\begin{array}{lcll} C, D, E & \longrightarrow & A \mid & \text{(primitive concept)} \\ & & \forall P.\, A \mid & \text{(typing of attribute)} \\ & & \exists P.\, \top \mid & \text{(necessary attribute)} \\ & & (\leq 1\, P) & \text{(single-valued attribute).} \end{array}$$

Schema axioms come in the two forms

$$A \sqsubseteq D, \qquad P \sqsubseteq A_1 \times A_2,$$

Construct	FOL Semantics	Set Semantics
$\forall P.\, A$	$\forall x.\, P(\gamma, x) \Rightarrow A(x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2.\, (d_1, d_2) \in P^{\mathcal{I}} \Rightarrow\ d_2 \in A^{\mathcal{I}}\}$
$\exists P.\, \top$	$\exists x.\, P(\gamma, x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2.\, (d_1, d_2) \in P^{\mathcal{I}}\}$
$(\leq 1\, P)$	$\forall x, y.\, P(\gamma, x) \wedge P(\gamma, y)$ $\Rightarrow x \doteq y$	$\left\{d_1 \in \Delta^{\mathcal{I}} \,\middle|\, \sharp\{d_2 \mid (d_1, d_2) \in P^{\mathcal{I}}\} \leq 1\right\}$
P^{-1}	$P(\beta, \alpha)$	$\{(d_2, d_1) \mid (d_1, d_2) \in P^{\mathcal{I}}\}$
$(R\colon C)$	$F_R(\alpha, \beta) \wedge F_C(\beta)$	$\{(d_1, d_2) \mid (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
ϵ	$\alpha \doteq \beta$	$\{(d_1, d_1) \mid d_1 \in \Delta^{\mathcal{I}}\}$
Qp	$\exists z.\, F_Q(\alpha, z) \wedge F_p(z, \beta)$	$\{(d_1, d_3) \mid \exists d_2.\, (d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_2, d_3) \in p^{\mathcal{I}}\}$
$\top$	**true**	$\Delta^{\mathcal{I}}$
$\{a\}$	$\gamma \doteq a$	$\{a^{\mathcal{I}}\}$
$C \sqcap D$	$F_C(\gamma) \wedge F_D(\gamma)$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\exists p$	$\exists x.\, F_p(\gamma, x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2.\, (d_1, d_2) \in p^{\mathcal{I}}\}$
$\exists p \doteq q$	$\exists x.\, F_p(\gamma, x) \wedge F_q(\gamma, x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2.\, (d_1, d_2) \in p^{\mathcal{I}} \wedge (d_1, d_2) \in q^{\mathcal{I}}\}$

Table 1: Transformational and set semantics of $\mathcal{SL}$ and $\mathcal{QL}$.

where A, A_1, A_2 are primitive concepts, D is an arbitrary $\mathcal{SL}$ concept, and P a primitive attribute. The first axiom states that all instances of A are instances of D. So D gives necessary conditions for membership in A. The second axiom states that the attribute P has domain A_1 and range A_2. An $\mathcal{SL}$ *schema* Σ consists of a set of schema axioms. As we will see below, by means of a schema we can represent attribute declarations and the clean part of class declarations.

In the *query language* $\mathcal{QL}$, attributes (ranged over by R) can be primitive attributes P or inverses P^{-1} of primitive attributes. Furthermore, there are *attribute restrictions*, written $(R\colon C)$, where R is an attribute and C is a $\mathcal{QL}$ concept. Intuitively, $(R\colon C)$ restricts the pairs related by R to those whose second component satisfies C. *Paths* (ranged over by p, q) are chains $(R_1\colon C_1)\cdots(R_n\colon C_n)$ of attribute restrictions and stand for the composition of the restricted attributes. The empty path is denoted as ϵ. In $\mathcal{QL}$, concepts are formed according to the rule:

$$
\begin{array}{lcll}
C, D, E & \longrightarrow & A \mid & \text{(primitive concept)} \\
 & & \top \mid & \text{(universal concept)} \\
 & & \{a\} \mid & \text{(singleton set)} \\
 & & C \sqcap D \mid & \text{(intersection)} \\
 & & \exists p \mid & \text{(existential quantification over path)} \\
 & & \exists p \doteq q & \text{(existential agreement of paths).}
\end{array}
$$

The intersection of concepts denotes the intersection of sets, the existential quantification over a path denotes those objects from which some object can be reached along the path, and the existential agreement of paths denotes those objects that have a common filler for the two paths. Observe that concepts and paths can be arbitrarily nested through attribute restrictions.

In Table 1 we present the semantics of attributes and concepts in two steps. Similar as in the previous section, we translate concepts, attributes, and paths into first order formulas (column 2). Then we give a semantics that treats concepts as set descriptions (column 3).

For the transformational semantics we map primitive concepts A and primitive roles P to atoms $A(\gamma)$ and $P(\alpha, \beta)$. Then column 2 contains for each complex concept C, attribute restriction Q and path p appearing in column 1 a corresponding formula $F_C(\gamma)$, $F_Q(\alpha, \beta)$ and $F_p(\alpha, \beta)$ that has one or two free variables, respectively.

$$
\begin{array}{lll}
\mathsf{Patient} \sqsubseteq \mathsf{Person} & \mathsf{Person} \sqsubseteq \forall \mathsf{name.\,String} & \mathsf{skilled\_in} \sqsubseteq \mathsf{Person} \times \mathsf{Topic} \\
\mathsf{Patient} \sqsubseteq \forall \mathsf{takes.\,Drug} & \mathsf{Person} \sqsubseteq \exists \mathsf{name.} \top & \\
\mathsf{Patient} \sqsubseteq \forall \mathsf{consults.\,Doctor} & \mathsf{Person} \sqsubseteq (\leq 1\, \mathsf{name}) & \\
\mathsf{Patient} \sqsubseteq \forall \mathsf{suffers.\,Disease} & & \\
\mathsf{Patient} \sqsubseteq \exists \mathsf{suffers.} \top & \mathsf{Doctor} \sqsubseteq \forall \mathsf{skilled\_in.\,Disease} &
\end{array}
$$

Figure 6: Schema axioms of the medical database

Given a fixed interpretation, each such formula denotes a binary or unary relation over the domain. Thus we can immediately formulate the semantics of attributes and concepts in terms of relations and sets without the detour through predicate logic notation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *extension function* of $\mathcal{I}$) that maps every concept to a subset of $\Delta^{\mathcal{I}}$, every constant to an element of $\Delta^{\mathcal{I}}$ and every attribute to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Moreover, we assume that distinct constants have distinct images (Unique Name Assumption). Given the denotation of primitive attributes and concepts, complex ones are interpreted according to the third column of Table 1 ("$\sharp\cdot$" denotes the cardinality of a set). It is easy to verify that column 3 just gives the sets of pairs and objects for which the formulas in column 2 hold.

We say that two concepts C, D are equivalent if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$, *i.e.*, equivalent concepts always describe the same sets.

We say that an interpretation $\mathcal{I}$ *satisfies* the axiom $A \sqsubseteq D$ if $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and the axiom $P \sqsubseteq A_1 \times A_2$ if $P^{\mathcal{I}} \subseteq A_1^{\mathcal{I}} \times A_2^{\mathcal{I}}$. If Σ is an $\mathcal{SL}$ schema, an interpretation $\mathcal{I}$ that satisfies all axioms in Σ is called a Σ-*interpretation*. A concept C is Σ-*satisfiable* if there is a Σ-interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. We say that C is Σ-*subsumed* by D (written $C \sqsubseteq_{\Sigma} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every Σ-interpretation $\mathcal{I}$.

3.2 The Concrete versus the Abstract

Next, we show by an example how to represent the clean part of a $\mathcal{DL}$ schema by a set of schema axioms and the clean part of a query class by a $\mathcal{QL}$ concept. Figure 6 gives the translation of our medical database schema from Figure 1 into schema axioms.

We demonstrate the transformation of query classes into $\mathcal{QL}$ concepts by translating as an example the query classes QueryPatient and ViewPatient from Subsection 2.2 into concepts C_Q and D_V. The inverses of attributes have to be made explicit. For instance, instead of using the attribute specialist one has to take skilled_in^{-1}.

$$
\begin{array}{rcl}
C_Q & = & \mathsf{Male} \sqcap \mathsf{Patient} \sqcap \\
& & \exists(\mathsf{consults\colon Female}) \doteq (\mathsf{suffers\colon} \top)(\mathsf{skilled\_in}^{-1}\mathsf{\colon Doctor}) \\
D_V & = & \mathsf{Patient} \sqcap \exists(\mathsf{name\colon String}) \sqcap \\
& & \exists(\mathsf{consults\colon Doctor})(\mathsf{skilled\_in\colon Disease}) \doteq (\mathsf{suffers\colon Disease}).
\end{array}
$$

Now we are able to reformulate our key problem in the new framework. Recall the question:

> *Given a $\mathcal{DL}$-schema $\mathcal{S}$, a query Q and a view V, are the instances of Q contained in the view V for every database state obeying the schema $\mathcal{S}$?*

Let Σ denote the translation of $\mathcal{S}$, C that of Q and D that of V. Since we forget about the dirty parts of $\mathcal{S}$, the restrictions for Σ-models are weaker than those for database states. Therefore a database state always corresponds to a Σ-model in a natural way. Since we forget about the dirty parts of Q, the answer set of Q is a subset of the denotation of C. So the instances of Q

D1: $F.G \rightarrow \{s\colon C_1,\ s\colon C_2\} \cup F.G$

if $s\colon C_1 \sqcap C_2$ is in F, and $s\colon C_1,\ s\colon C_2$ are not both in F

D2: $F.G \rightarrow \{sRt\} \cup F.G$

if $tR^{-1}s$ is in F, and sRt is not in F

D3: $F.G \rightarrow (F.G)[y/a]$

if $y\colon \{a\}$ is in F

D4: $F.G \rightarrow \{spy\} \cup F.G$

if $s\colon \exists p$ is in F, and there is no y' with spy' in F, and y is a fresh variable

D5: $F.G \rightarrow \{sps\} \cup F.G$

if $s\colon \exists p \doteq \epsilon$ is in F, and sps is not in F

D6: $F.G \rightarrow \{sRy, y\colon C, ypt\} \cup F.G$

if $s(R\colon C)pt$ is in F, and there is no y' such that sRy', $y\colon' C$, $y'pt$ are all in F, and $p \neq \epsilon$, and y is fresh variable

D7: $F.G \rightarrow \{sRt, t\colon C\} \cup F.G$

if $s(R\colon C)t$ is in F, and sRt, $t\colon C$ are not both in F

Figure 7: The decomposition rules

are surely contained in V if C is Σ-subsumed by D (recall that a view has no dirty part and thus is entirely captured by the concept).

Proposition 3.1 *Let $\mathcal{S}$, Q, V be a schema, a query class, and a view in $\mathcal{DL}$, and let Σ, C, D be their translations into an $\mathcal{SL}$ schema and $\mathcal{QL}$ concepts. If*

$$C \sqsubseteq_\Sigma D$$

then in every state of $\mathcal{S}$, the answer set of Q is a subset of the answer set of V.

The "only if" direction does not hold, since we forget about dirty parts. So Σ-subsumption gives us a sufficient but not a necessary condition for subsumption of queries. In the next section we will present a calculus for detecting Σ-subsumption, which provides a procedure that runs in time polynomial in the size of schema, query and view.

4 A Calculus for Deciding Subsumption

The basic idea for deciding subsumption between a query concept C and a view concept D is as follows. We take an object o and transform C into a prototypical database state where o is an instance of C. We do so by generating objects, entering them into concepts, and relating them through attributes. Then we evaluate D against this database. If o belongs to the answers of D then C is subsumed by D. If not, we have a state where an object is in C but not in D and therefore C is not subsumed by D.

This approach is similar to the technique used for deciding inclusion of conjunctive queries (see [Ull89]). But the problem is more complicated in our case because we have to take into account the schema axioms. In particular, axioms of the form $A \sqsubseteq \exists R.\top$ lead to complications, since they can enforce the generation of new objects. To guarantee that the database is of polynomial size and that D can be evaluated in polynomial time, D is used to provide guidance for the construction of the database.

S1: $F.G \rightarrow \{s{:}A_2\} \cup F.G$

if $s{:}A_1$ is in F, $A_1 \sqsubseteq A_2$ is in Σ, and $s{:}A_2$ is not in F

S2: $F.G \rightarrow \{t{:}A_2\} \cup F.G$

if $s{:}A_1$ and sPt are in F, and $A_1 \sqsubseteq \forall P.\, A_2$ is in Σ, and $t{:}A_2$ is not in F

S3: $F.G \rightarrow \{s{:}A_1,\ t{:}A_2\} \cup F.G$

if sPt is in F, and $P \sqsubseteq A_1 \times A_2$ is in Σ, and $s{:}A_1$, $t{:}A_2$ are not both in F

S4: $F.G \rightarrow (F.G)[y/t]$

if $s{:}A$, sPy, sPt are in F, and $A \sqsubseteq (\leq 1\ P)$ is in Σ

S5: $F.G \rightarrow \{sPy\} \cup F.G$

if $s{:}\exists(P{:}C)p$ is in G or $s{:}\exists(P{:}C)p \doteq \epsilon$ is in G, and there is no y' with sPy' in F, and there is an A with $s{:}A$ in F and $A \sqsubseteq \exists P.\, \top$ in Σ, and y is a fresh variable

Figure 8: The schema rules

Any concept of the form $\exists p \doteq q$ is equivalent to a concept of the form $\exists p' \doteq \epsilon$, since paths can be inverted using inverses of attributes. In the sequel we assume that no concept has subconcepts of the form $\exists p \doteq q$ where $q \neq \epsilon$, since this simplifies the calculus.

4.1 The Rules of the Calculus

To formulate the calculus we augment our syntax by *variables* (ranged over by x, y). We will refer to constants and variables alike as *individuals* (denoted by the letters s, t). Our calculus works on syntactic entities called *constraints*[6] that have one of the forms

$$s{:}C, \qquad sRt, \qquad spt,$$

where C is a $\mathcal{QL}$ concept, R is an attribute, p is a path, and s, t are individuals. The first constraint says that s is an instance of C, the second that t is an R-filler of s, and the third that s and t are related through p. A *constraint system* is a set S of constraints.

We also extend the semantics. An interpretation $\mathcal{I}$ maps a variable x to an element $x^\mathcal{I}$ of its domain. It *satisfies* a constraint $s{:}C$ if $s^\mathcal{I} \in C^\mathcal{I}$, a constraint sRt if $(s^\mathcal{I}, t^\mathcal{I}) \in R^\mathcal{I}$, and a constraint spt if $(s^\mathcal{I}, t^\mathcal{I}) \in p^\mathcal{I}$. We say that a Σ-interpretation $\mathcal{I}$ is a Σ-*model* of a constraint c if it satisfies c. A constraint is Σ-*satisfiable* if it has a Σ-model. The notions of satisfaction, model, and satisfiability are extended to constraint systems as one would expect.

[6]Not to be confused with the constraints in class declarations!

G1: $F.G \rightarrow F.G \cup \{s{:}C, s{:}D\}$

if $s{:}C \sqcap D$ is in G, and $s{:}C$ and $s{:}D$ are not in $G \cup F$

G2: $F.G \rightarrow F.G \cup \{t{:}C\}$

if $s{:}\exists(R{:}C)$ is in G or $s{:}\exists(R{:}C) \doteq \epsilon$ is in G, and sRt is in F, and $t{:}C$ is not in $G \cup F$

G3: $F.G \rightarrow F.G \cup \{t{:}C, t{:}\exists p\}$

if $s{:}\exists(R{:}C)p$ is in G or $s{:}\exists(R{:}C)p \doteq \epsilon$ is in G, and sRt is in F, and $t{:}C$, $t{:}\exists p$ are not all in $G \cup F$

Figure 9: The goal rules

C1: $F.G \rightarrow \{s{:}\, C \sqcap D\} \cup F.G$

if $s{:}\, C$ and $s{:}\, D$ are in F, and $s{:}\, C \sqcap D$ is in G but not in F

C2: $F.G \rightarrow \{s{:}\, \top\} \cup F.G$

if $s{:}\, \top$ is in G but not in F

C3: $F.G \rightarrow \{s{:}\, \exists p\} \cup F.G$

if $p = \epsilon$ or there is a t with spt in F, and $s{:}\, \exists p$ is in G but not in F

C4: $F.G \rightarrow \{s{:}\, \exists p \doteq \epsilon\} \cup F.G$

if $p = \epsilon$ or sps is in F, and $s{:}\, \exists p \doteq \epsilon$ is in G but not in F

C5: $F.G \rightarrow \{s(R{:}\, C)pt\} \cup F.G$

if there is a t' with sRt', $t'{:}\, C$ and $t'pt$ in F and $s{:}\, \exists(R{:}\, C)p$ or $s{:}\, \exists(R{:}\, C)p \doteq \epsilon$ is in G, and $s(R{:}\, C)pt$ is not in F

C6: $F.G \rightarrow \{s(R{:}\, C)t\} \cup F.G$

if sRt and $t{:}\, C$ are in F, and $s{:}\, \exists(R{:}\, C)$ or $s{:}\, \exists(R{:}\, C) \doteq \epsilon$ is in G, and $s(R{:}\, C)t$ is not in F

Figure 10: The composition rules

Let c, c' be constraints and Σ be an $\mathcal{SL}$ schema. We write

$$c \models_\Sigma c'$$

if every Σ-model of c is also a Σ-model of c'. This notion of Σ-entailment is naturally extended to constraint systems. The following proposition describes how Σ-entailment is linked to Σ-subsumption.

Proposition 4.1 *Let Σ be an $\mathcal{SL}$ schema, C, D be $\mathcal{QL}$ concepts, and x be a variable. Then*

$$C \sqsubseteq_\Sigma D \quad \textit{iff} \quad x{:}\, C \models_\Sigma x{:}\, D.$$

General Assumption. *Throughout this section, Σ denotes a fixed $\mathcal{SL}$ schema.*

Our calculus features four kinds of rules: decomposition, schema, goal, and composition rules. The rules work on pairs of constraint systems $F.G$ (called *pairs* for short). We call F the *facts* and G the *goals*. In order to decide whether $C \sqsubseteq_\Sigma D$, we take a variable x and start with the fact $\{x{:}\, C\}$ and the goal $\{x{:}\, D\}$. The rules will add more facts and goals until no more rule is applicable. Intuitively, the query C is subsumed by the view D iff the final set of facts contains the constraint $x{:}\, D$.[7] All rules exploit the hierarchical structure of concepts, which is the basic reason for the polynomiality of the procedure.

To formulate the rules we use the following notation. By R^{-1} we denote P^{-1} if $R = P$ and P if $R = P^{-1}$. The pair $(F.G)[y/s]$ is obtained from $F.G$ by replacing every occurrence of y with s.

The rules are presented in Figures 7 to 10. The *decomposition rules* (Figure 7) work on facts. They break up the initial fact $x{:}\, C$ into constraints involving only primitive concepts, primitive attributes, and singletons. In breaking up a path, Rules D4 and D6 use fresh variables to represent the objects along the path.

The *schema rules* (Figure 8) also work on facts. They add information derivable from the schema and the current facts. The first four rules are simple: Rules S1 to S3 add membership constraints for individuals in F, and Rule S4 identifies values of functional attributes. Rule

[7] We will see later on that this condition has to be refined a bit.

$$
\begin{array}{lcll}
F_2 & = & F_1 \cup \{x\colon \textsf{Male} \sqcap \textsf{Patient}, & \\
 & & \quad x\colon \exists(\textsf{consults}\colon \textsf{Female} \sqcap \textsf{Doctor})(\textsf{skilled\_in}\colon \top)(\textsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \text{D1} \\
F_3 & = & F_2 \cup \{x\colon \textsf{Male}, x\colon \textsf{Patient}\} & \text{D1} \\
F_4 & = & F_3 \cup \{x\, (\textsf{consults}\colon \textsf{Female} \sqcap \textsf{Doctor})(\textsf{skilled\_in}\colon \top)(\textsf{suffers}^{-1}\colon \top)\, x\} & \text{D5} \\
F_5 & = & F_4 \cup \{x \,\textsf{consults}\, s_1, y_1\colon \textsf{Female} \sqcap \textsf{Doctor}, y_1\, (\textsf{skilled\_in}\colon \top)(\textsf{suffers}^{-1}\colon \top)\, x\} & \text{D6} \\
F_6 & = & F_5 \cup \{y_1\colon \textsf{Female}, y_1\colon \textsf{Doctor}\} & \text{D1} \\
F_7 & = & F_6 \cup \{y_1 \,\textsf{skilled\_in}\, y_2, y_2\colon \top, y_2\, (\textsf{suffers}^{-1}\colon \top)\, x\} & \text{D6} \\
F_8 & = & F_7 \cup \{y_2 \,\textsf{suffers}^{-1}\, x, x\colon \top\} & \text{D7} \\
F_9 & = & F_8 \cup \{x \,\textsf{suffers}\, y_2, y_1 \,\textsf{consults}^{-1}\, x, y_2 \,\textsf{skilled\_in}^{-1}\, y_1\} & 3\times\text{D2} \\
F_{10} & = & F_9 \cup \{x\colon \textsf{Person}\} & \text{S1} \\
F_{11} & = & F_{10} \cup \{y_2\colon \textsf{Disease}\} & \text{S2} \\
F_{12} & = & F_{11} \cup \{y_1\colon \textsf{Person}, y_2\colon \textsf{Topic}\} & \text{S3} \\
G_2 & = & G_1 \cup \{x\colon \textsf{Patient} \sqcap \exists(\textsf{name}\colon \textsf{String}), & \\
 & & \quad x\colon \exists(\textsf{consults}\colon \textsf{Doctor})(\textsf{skilled\_in}\colon \textsf{Disease})(\textsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \text{G1} \\
G_3 & = & G_2 \cup \{x\colon \textsf{Patient}, x\colon \exists(\textsf{name}\colon \textsf{String})\} & \text{G1} \\
G_4 & = & G_3 \cup \{y_1\colon \textsf{Doctor}, y_1\colon \exists(\textsf{skilled\_in}\colon \textsf{Disease})(\textsf{suffers}^{-1}\colon \top)\} & \text{G3} \\
G_5 & = & G_4 \cup \{y_2\colon \exists(\textsf{suffers}^{-1}\colon \top)\} & \text{G3} \\
F_{13} & = & F_{12} \cup \{x \,\textsf{name}\, y_3\} & \text{S5} \\
F_{14} & = & F_{13} \cup \{y_3\colon \textsf{String}\} & \text{S2} \\
F_{15} & = & F_{14} \cup \{y_3 \,\textsf{name}^{-1}\, x\} & \text{D2} \\
F_{16} & = & F_{15} \cup \{y_1\, (\textsf{skilled\_in}\colon \textsf{Disease})(\textsf{suffers}^{-1}\colon \top)\, x\} & \text{C5} \\
F_{17} & = & F_{61} \cup \{x\, (\textsf{consults}\colon \textsf{Doctor})(\textsf{skilled\_in}\colon \textsf{Disease})(\textsf{suffers}^{-1}\colon \top)\, x\} & \text{C5} \\
F_{18} & = & F_{17} \cup \{x\colon \exists(\textsf{consults}\colon \textsf{Doctor})(\textsf{skilled\_in}\colon \textsf{Disease})(\textsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \text{C4} \\
F_{19} & = & F_{18} \cup \{x\colon \exists(\textsf{name}\colon \textsf{String})\} & \text{C6} \\
F_{20} & = & F_{19} \cup \{x\colon \textsf{Patient} \sqcap \exists(\textsf{name}\colon \textsf{String})\} & \text{C1} \\
F_{21} & = & F_{20} \cup \{x\colon \textsf{Patient} \sqcap \exists(\textsf{name}\colon \textsf{String}) \sqcap & \\
 & & \quad \exists(\textsf{consults}\colon \textsf{Doctor})(\textsf{skilled\_in}\colon \textsf{Disease})(\textsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \text{C1}
\end{array}
$$

Figure 11: A sequence of rule applications

S5, however, which might create a new individual, is subject to a tricky control that limits the number of new individuals: it is only applicable if it contributes to a path that is required by a goal.

Also individuals introduced by the decomposition rules can help in building up such a path. Since they carry more specific information than variables created by schema rules, decomposition rules receive priority:

- A schema rule can be applied only if no decomposition rule is applicable.

This control structure contributes to keeping the whole procedure polynomial.

The *goal rules* (Figure 9) work on goals. They guide the evaluation of the view concept D by deriving subgoals from the original goal $x\colon D$. The interesting rules are G2 and G3, since they relate goals to facts: if the goal is to find a path issuing from s whose first step involves the attribute R, then only individuals t are tested which are explicitly mentioned as R-fillers of s in the facts.

The *composition rules* (Figure 10) compose complex facts from simpler ones directed by the goals. This can be understood as a bottom up evaluation of concept D over F.

A pair is *complete* if no rule is applicable. A complete pair obtained from a pair $F.G$ by applying the above rules is called a *completion* of $F.G$. Since all rules are deterministic, there exists—up to variable renaming—exactly one completion for a pair of constraint systems.

As an example, we use the calculus to check the concepts C_Q and D_V of Section 3.2 for subsumption. We start with the pair $F_1 . G_1$ where:

$$F_1 = \{x\colon \mathsf{Male} \sqcap \mathsf{Patient} \sqcap \exists(\mathsf{consults}\colon \mathsf{Female} \sqcap \mathsf{Doctor})(\mathsf{skilled\_in}\colon \top)(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\}$$

$$G_1 = \{x\colon \mathsf{Patient} \sqcap \exists(\mathsf{name}\colon \mathsf{String}) \sqcap \exists(\mathsf{consults}\colon \mathsf{Doctor})(\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\}.$$

Note that C_Q and D_V have been rewritten such that all agreements of paths have the form $\exists p \doteq \epsilon$. Figure 11 shows a sequence of rule applications. In each step we give only the component of the pair that changes. As completion we obtain $F_{21} . G_5$ and see that $x\colon D_V$ is in F_{21}. Hence $C_Q \sqsubseteq_\Sigma D_V$.

Not every constraint system is Σ-satisfiable. Since different constants are interpreted as distinct objects, a constraint of the form $a\colon \{b\}$ is unsatisfiable. For the same reason, different constants cannot be values of a functional attribute at the same time. These observations are captured by the notion of clash.

A *clash* is a constraint system of one of the following forms:

- $\{a\colon \{b\}\}$, where $a \neq b$
- $\{sPa, sPb, s\colon A\}$, where $A \sqsubseteq (\leq 1\ P)$ is in Σ and $a \neq b$.

A constraint system is *clash-free* if it does not contain a clash. Obviously, a constraint system containing a clash is not Σ-satisfiable. We will show later on that a clash-free set of facts in a complete pair is Σ-satisfiable.

4.2 Soundness and Completeness of the Calculus

We now show that our calculus indeed gives rise to a polynomial time decision procedure for Σ-subsumption of $\mathcal{QL}$ concepts.

First, notice that in any pair $F . G$ derivable from an initial pair $\{x\colon C\} . \{x\colon D\}$, the set of goals G contains exactly one constraint of the form $s\colon D$. Moreover, if $s\colon D \in G$, then $s\colon C \in F$. This can be verified by inspecting the rules that alter goals.

General Assumption. *In the sequel of this section, C, D are $\mathcal{QL}$ concepts, x is a variable, $F_C . G_D$ is the completion of $\{x\colon C\} . \{x\colon D\}$, and o is the individual such that $o\colon D$ is in G_D.*

We first observe that the calculus adds only consequences to the set of facts, *i.e.*, it keeps the models of the facts invariant.

Proposition 4.2 (Invariance) *Suppose $F . G$ has been derived from $\{x\colon C\} . \{x\colon D\}$, and $F' . G'$ is obtained from $F . G$ by applying a rule. Then every Σ-model $\mathcal{I}$ of F can be turned into a Σ-model $\mathcal{I}'$ of F' by modifying the interpretation of fresh variables. Moreover, if s, s' are such that $\{s\colon D\} \in G$ and $\{s'\colon D\} \in G'$, then $\mathcal{I}'$ can be chosen such that $s'^{\mathcal{I}'} = s^{\mathcal{I}}$.*

Proof. The claim can be shown by a case analysis considering all rules. □

Corollary 4.3 *Every Σ-model $\mathcal{I}$ of $x\colon C$ can be turned into a Σ-model $\mathcal{I}'$ of F_C by modifying the interpretation of variables. Moreover, $\mathcal{I}'$ can be chosen such that $o^{\mathcal{I}'} = x^{\mathcal{I}}$.*

Proof. This follows by induction from the preceding proposition. □

Corollary 4.4

$$x\colon C \models_\Sigma x\colon D \quad \Longleftrightarrow \quad F_C \models_\Sigma o\colon D$$

Proof. "$\Rightarrow$" Let $\mathcal{I}$ be a Σ-model of F_C. Then $\mathcal{I}$ is a Σ-model of $o\colon C$, since $o\colon C \in F_C$. Let $\mathcal{I}'$ be such that $x^{\mathcal{I}'} = o^{\mathcal{I}}$ and $\mathcal{I}'$ coincides with $\mathcal{I}$ otherwise. Then $\mathcal{I}'$ is a Σ-model of $x\colon C$ and hence of $x\colon D$. By definition, $\mathcal{I}'$ is a Σ-model of $o\colon D$ as well. Since $\mathcal{I}$ and $\mathcal{I}'$ coincide on all symbols, except possibly x, it follows that $\mathcal{I}$ is a Σ-model of $o\colon D$.

"$\Leftarrow$" Let $\mathcal{I}$ be a Σ-model of $x\colon C$. By the preceding corollary, $\mathcal{I}$ can be turned into a Σ-model $\mathcal{I}'$ of F_C with $o^{\mathcal{I}'} = x^{\mathcal{I}}$ by modifying the interpretation of variables. The interpretation $\mathcal{I}'$ is also a Σ-model of $o\colon D$. Since $D^{\mathcal{I}} = D^{\mathcal{I}'}$ and $x^{\mathcal{I}} = o^{\mathcal{I}'}$ it follows that $\mathcal{I}$ is a Σ-model of $x\colon D$. $\square$

From constraint systems one can derive interpretations in a natural way. Let u be a new symbol and S be a constraint system. The *canonical interpretation* $\mathcal{I}_S$ of S is defined as follows:

$$\begin{aligned}
\Delta^{\mathcal{I}_S} &:= \{s \mid s \text{ is an individual in } S\} \cup \{u\} \\
s^{\mathcal{I}_S} &:= s \\
A^{\mathcal{I}_S} &:= \{s \mid s\colon A \text{ is in } S\} \cup \{u\} \\
P^{\mathcal{I}_S} &:= \{(s,t) \mid sPt \text{ is in } S\} \cup \{(u,u)\} \\
&\quad\; \cup \{(s,u) \mid \text{there is no } sPt \text{ in } S, \text{ but for some } A, \\
&\qquad\qquad\qquad s\colon A \text{ is in } S \text{ and } A \sqsubseteq \exists P.\top \text{ is in } \Sigma\}.
\end{aligned}$$

We will be particularly interested in canonical interpretations that are obtained from the facts of a complete pair. As pointed out before, the schema rules are designed in such a way that not every necessary attribute will get a variable as a filler. The role of the new object u is to compensate for this lack.

The following proposition makes the idea more precise that our calculus constructs a Σ-model of C.

Proposition 4.5 *Let $F.G$ be a complete pair. If F is clash-free, then the canonical interpretation $\mathcal{I}_F$ is a Σ-model of F.*

Proof. We have to verify that $\mathcal{I}_F$ satisfies every axiom in Σ and every constraint in F.

We first consider the schema axioms. Suppose that Σ contains an axiom of the form $A \sqsubseteq \exists P.\top$. Let $s \in A^{\mathcal{I}_F}$. If $s = u$, then the axiom is satisfied, since $(u,u) \in P^{\mathcal{I}_F}$. If $s \neq u$, then $s\colon A \in F$. We distinguish two cases: (1) there is a constraint sPt in F, or (2) there is no such constraint. In the first case, $(s,t) \in P^{\mathcal{I}_F}$ and in the second, $(s,u) \in P^{\mathcal{I}_F}$. Thus the axiom is satisfied. The other cases require similar reasoning and are therefore dismissed.

Next we consider the different constraints. By definition of $\mathcal{I}_F$, every constraint sPt is satisfied. If F contains a constraint $sP^{-1}t$, then it contains also the constraint tPs, since otherwise Rule D2 would be applicable. Obviously, every constraint $s\colon A$ and $s\colon \top$ is satisfied.

To prove that more complex constraints are satisfied, we proceed by induction. If F contains $s\colon E \sqcap E'$, then, because of Rule D1, it contains as well $s\colon E$ and $s\colon E'$, which are satisfied by the inductive hypothesis. Hence, $\mathcal{I}_F$ satisfies also $s\colon E \sqcap E'$. If F contains $s(R\colon C)t$, then, because of Rule D7, it contains as well sRt and $t\colon C$, which are satisfied by the inductive hypothesis. Hence, $\mathcal{I}_F$ satisfies also $s(R\colon C)t$. We drop the remaining cases, since the arguments will be similar. $\square$

Proposition 4.6 *Let $\mathcal{I}_{F_C}$ be the canonical interpretation of F_C and $s\colon E$ be a constraint in G_D. If F_C is clash-free, then*

$$\mathcal{I}_{F_C} \text{ satisfies } s\colon E \quad \Longrightarrow \quad s\colon E \in F_C.$$

Proof. Suppose that F_C is clash-free and that $\mathcal{I}_{F_C}$ satisfies $s\colon E$. The proof is by induction on the structure of E.

We first consider the base cases. Suppose that $E = A$. Then $s{:}\, A \in F_C$ by definition of $\mathcal{I}_{F_C}$, since $\mathcal{I}_{F_C}$ satisfies $s{:}\, A$. Suppose that $E = \top$. Then $s{:}\, \top \in F_C$ because of Rule C2. Suppose that $E = \{a\}$. In order to deal with this case, we have to make use of two observations. First, noticing the fact that our calculus starts off with a pair $\{x{:}\, C\}.\{x{:}\, D\}$, and inspecting the rules of the calculus we see that any individual t occurring in a constraint $t{:}\, E'$ in G_D occurs also in F_C. Second, analyzing the rules again we see that if a constant a occurs in F_C, then F_C contains a constraint $t{:}\, \{a\}$. Now, if $\mathcal{I}_{F_C}$ satisfies $s{:}\, \{a\}$, then by definition of $\mathcal{I}_{F_C}$ we have $s{:} = a$. By our first observation, a occurs also in F_C, and by our second observation, F_C contains a constraint $t{:}\, \{a\}$. Since F_C is the first component of a complete pair, it follows that $t = a$ and hence that $a{:}\, \{a\} \in F_C$.

Next we analyze in turn each possible complex concept E. If E is of the form $E' \sqcap E''$ then $\mathcal{I}_{F_C}$ satisfies $s{:}\, E' \sqcap E''$ iff $\mathcal{I}_{F_Q}$ satisfies $s{:}\, E'$ and $s{:}\, E''$. By the inductive hypothesis, this is the case iff $s{:}\, E' \in F_C$ and $s{:}\, E'' \in F_C$. Since $s{:}\, E' \sqcap E'' \in G_D$, we have that $s{:}\, E' \sqcap E'' \in F_C$, since otherwise Rule C1 would be applicable.

If E is of the form $\exists p$ then we have to consider two cases. The case $p = \epsilon$ is captured by Rule C3. If $p \neq \epsilon$, then $\mathcal{I}_{F_C}$ satisfies $s{:}\, \exists p$ iff $\mathcal{I}_{F_C}$ satisfies spt for an individual t. By an induction on the length of p, which is nested in the current induction, we show that $spt \in F_C$ if $\mathcal{I}_{F_C}$ satisfies spt. Then, because of Rule C3, $s{:}\, \exists p \in F_C$.

As the base case of the nested induction we consider $p = (R{:}\, E')$. Then, since $s{:}\, \exists(R{:}\, E') \in G_D$ and since $F_C.G_D$ is complete, by the definition of $\mathcal{I}_{F_C}$ it follows that $\mathcal{I}_{F_C}$ satisfies $s(R{:}\, E')t$ iff $sRt \in F_C$ and $\mathcal{I}_{F_C}$ satisfies $t{:}\, E'$. By the outer inductive hypothesis this is the case if $t{:}\, E' \in F_C$ (since $t{:}\, E' \in G_D$). Then, because of Rule C6, we have $spt \in F_C$.

For the induction step suppose that $p = (R{:}\, E')p'$. Since $s{:}\, \exists(R{:}\, E')p' \in G_D$ and since $F_C.G_D$ is complete, by the definition of $\mathcal{I}_{F_C}$ it follows that $\mathcal{I}_{F_C}$ satisfies $s(R{:}\, E')p't$ iff there is a t' such that $sRt' \in F_C$ and $\mathcal{I}_{F_C}$ satisfies both $t'{:}\, E'$ and $t'p't$. By the inner and the outer inductive hypothesis this is the case if $t'{:}\, E' \in F_C$ and $t'p't \in F_C$ (since $t'{:}\, E' \in G_D$ and $t'{:}\, \exists p' \in G_D$). Then, because of Rule C5, we have $spt \in F_D$, which ends the inner induction proof.

If E is of the form $\exists p \doteq \epsilon$ then we again have to consider two cases. The case $p = \epsilon$ is captured by Rule C4. If $p \neq \epsilon$, then $\mathcal{I}_{F_C}$ satisfies $s{:}\, \exists p \doteq \epsilon$ iff $\mathcal{I}_{F_C}$ satisfies sps. If $\mathcal{I}_{F_C}$ satisfies sps, then sps is in F_C. This can be shown by an induction analogous to the one above. Then because of Rule C4, we have $s{:}\, \exists p \doteq \epsilon \in F_C$. □

Now, we only have to put together the previous statements to show the soundness and completeness of the calculus.

Theorem 4.7 (Soundness and Completeness)

$$C \sqsubseteq_\Sigma D \quad \Longleftrightarrow \quad o{:}\, D \in F_C \ \textit{ or } \ F_C \textit{ contains a clash.}$$

Proof. By Proposition 4.1, we have $C \sqsubseteq_\Sigma D$ if and only if $x{:}\, C \models_\Sigma x{:}\, D$.

"$\Rightarrow$" Suppose $x{:}\, C \models_\Sigma x{:}\, D$ and F_C does not contain a clash. By Corollary 4.4, we know that $F_C \models_\Sigma o{:}\, D$. Since F_C is clash-free, Proposition 4.5 implies that $\mathcal{I}_{F_C}$ is a model of F_C and hence, $\mathcal{I}_{F_C}$ satisfies $o{:}\, D$. Since clearly $o{:}\, D$ is in G_D, we conclude by Proposition 4.6 that $o{:}\, D \in F_C$.

"$\Leftarrow$" If $o{:}\, D \in F_C$, then $F_C \models_\Sigma o{:}\, D$, and Corollary 4.4 yields the claim. If F_C contains a clash, then F_C is unsatisfiable and hence, $x{:}\, C$ is unsatisfiable too, by Corollary 4.3. Thus, C is unsatisfiable and therefore Σ-subsumed by any concept. □

Now we turn to the complexity of deciding Σ-subsumption. To this end we estimate the number of individuals in $F_C.G_D$. First, observe that any constant in this pair must appear in C. Second, for every variable introduced by a decomposition rule, there is an existentially quantified subconcept of C. Hence, the number of constants plus the number of variables generated by decomposition rules is less or equal to the size of C. We call these individuals *primary individuals*. Moreover, we say that t' is a *successor* of t if F_C contains constraints $tP_1s_1,\ s_1P_2s_2,\ \ldots, s_{j-1}P_jt'$. Third, since the introduction of variables by the schema rule S5 is controlled by the structure of D, one can show that for every primary individual the number of nonprimary successors is bounded by the size of D. Summarizing, we obtain an upper bound for the number of individuals occurring in $F_C.G_D$.

Proposition 4.8 *The number of individuals occurring in $F_C.G_D$ is at most MN, where M is the size of C and N is the size of D.*

Every application of a rule either adds new constraints or reduces the number of variables. Since each new constraint is a combination of variables and concept or attribute expressions occurring in C, D, or Σ, the number of rule applications adding constraints is polynomially bounded by the size of C, D and Σ. Similarly, the number of rule applications that identify individuals is bounded by the number of individuals generated through the computation. Summarizing, we conclude that any derivation of $F_C.G_D$ comprises polynomially many steps. Moreover, testing whether a rule is applicable and applying a rule can be accomplished in time polynomial in the size of Σ and the current constraint system. Thus, a completion of $\{x: C\}.\{x: D\}$ can be computed in time polynomial in the size of C, D and Σ. This yields our second theorem.

Theorem 4.9 *Σ-subsumption between $\mathcal{QL}$ concepts can be decided in time polynomial in the size of C, D and Σ.*

Polynomiality is lost if we allow inverses of attributes in the schema language. Then deciding Σ-subsumption between primitive concepts is already NP-hard. Adding constructs to $\mathcal{SL}$ that allow one to specify schemas with unsatisfiable primitive concepts, *e.g.*, singletons or negation of primitive concepts, leads also to NP-hardness of Σ-subsumption. In this sense our languages are designed so as to gain maximal expressiveness without loosing tractability.

The presented algorithm can be embedded into a query optimizer for $\mathcal{DL}$ database systems: instead of just employing conventional compilation techniques for generating query evaluation plans from query classes, a subsumption checking component tests whether an incoming query is subsumed by one of the views currently materialized in the database. For this purpose the clean parts of query classes and the view definitions are translated into $\mathcal{QL}$ expressions, the schema declarations into $\mathcal{SL}$ expressions, and our algorithm is run as explained above. The system modifies the query evaluation plans by adding access operations to the stored extensions of subsuming views, thus restricting the search space. We plan to implement such a subsumption checker within the deductive object base manager *ConceptBase* [JS93, SNJ93] which offers a schema and query language very similar to $\mathcal{DL}$.

5 Related Work

Our work relates to several fields of research in databases and AI. We shortly discuss the relationship to optimization of conjunctive queries, to semantic query optimization, and query optimization in existing OODB systems.

Conjunctive Queries. A closer look at Table 1 reveals that $\mathcal{QL}$ concepts are equivalent to formulas, whose prefix has only existential quantifiers and whose matrix is a conjunction of positive function free atoms. Such formulas are known as *conjunctive queries* in the context of relational and deductive databases [Ull89]. Much effort has been devoted to the *containment* problem for such queries, *i.e.*, subsumption between queries with respect to an empty schema. However, the objective of this work was not reusing queries, but computing for a given query an equivalent one by removing unnecessary conjuncts. It is known that deciding containment of conjunctive queries is NP-hard, even if all predicates involved are binary. Aho *et al.* [ASU79] and Johnson and Klug [JK83] identified classes of conjunctive queries for which subsumption can be decided in polynomial time. The class described by Aho *et al.* is so restricted that even queries that chain one attribute (like chaining "child" so as to yield "grandchild") are not captured, while the queries studied by Johnson and Klug are defined by means of complicated graph theoretic properties. Neither of these classes comprises the language $\mathcal{QL}$, so that $\mathcal{QL}$ concepts can be seen as a naturally occurring class of conjunctive queries with polynomial containment problem. In all this work on conjunctive queries no schema information like in our case has been taken into account. Recently, Chan [Cha92] has investigated the optimization of conjunctive queries in an object-oriented setting. He considered some minimal schema information like subclass relationship and disjointness of classes. Although the containment problem for his language is obviously NP-hard, he did not address the question of complexity.

Semantic Query Optimization. Semantic query optimization considers semantic knowledge (integrity constraints) for constructing query evaluation plans and is relevant due to the incorporation of schema knowledge in our algorithm. Semantic optimization techniques were first proposed in [Kin81, HZ80, Jar84] in the context of relational databases and dealt with rather simple types of constraints stating *e.g.*, referential integrity and functional dependencies. For deductive databases and general integrity constraints in clausal form, [CGM88, CGM90, GL92, Kow92] describe a rewriting of queries and rules. This technique has also been used for generating cooperative answers [Gaa92]. Several papers [SO89, HLO91] deal with the implementation of semantic query optimizers, esp. schemes for deciding which rules and integrity constraints are actually promising profit for a given query. Within this framework, our method belongs to the category of approaches that respect constraints (in the database schema) of specific types, namely value restrictions, existential and functional requirements for attribute fillers and subset relationships. Obviously, there is a direct correspondence between the expressiveness of the language $\mathcal{SL}$ and the handled classes of constraints.

Object-oriented Query Optimization. Object-oriented database systems like O_2 [OT92, BCD92] and ObjectStore [OHMS92] focus query optimization on the use of physical clustering strategies and indexes. In O_2, indexes are quite flexible by allowing a membership condition and computed attributes specified as path expressions, similar to the role of views presented in this paper. However, the schema of the O_2 database is not taken into account for query optimization. ObjectStore concentrates on indexes for path expressions which allows an easier selection of indexes for a given query. Both OODB systems do not provide automatic maintenance of their indexes. In O_2, the application program must take care of it, and in ObjectStore the database designer has to annotate the schema. Our approach makes this overhead unnecessary since the triggers for view maintenance can be automatically generated from the logical representation of views. Path expressions, derived attributes, views, and membership conditions to views are very similar to XSQL [KKS92]. Our notion of queries and views extends XSQL by class membership constraints to each intermediate node in the path. On the other hand, XSQL provides generation of object identifiers which is not considered here. Specific OODB views called materialized functions have been investigated in [KMWZ91]. The functions are used for deriving attribute values which are stored in a separate data structure. A similar strategy is pursued in the extended relational database system Postgres [SK91]. Cost models and benchmark resuls are in [KKM91] and [Han87] (for relational databases).

There are several other systems combining advantages of reasoning in concept languages with object-oriented (*e.g.*, [SLT91, LRS92]) or rule-based (*e.g.*, [BHHM91]) representation languages by a hybrid system architecture. The idea to use different concept languages for stating facts and for querying has also appeared in [LS91].

6 Conclusions

This paper proposed a query optimization technique for OODB's by exploiting the structural knowledge encoded in the schema and in the query definitions. Summarizing we used the following ideas:

1. Queries and class definitions of OODB's are separated into a *clean* part (structural membership condition) and a *dirty* part (remaining membership condition). Query optimization by subsumption is restricted to the clean part which guarantees polynomial efficiency.
2. Our algorithm incorporates knowledge from the schema level for finding additional subsumptions which are not derivable from query definitions only. Particularly, the necessary category for attributes, a common existential constraint, is employed for deriving non-trivial subsumptions.
3. The correspondence between concept languages and the first-order semantics of OODB's makes view maintenance methods from deductive databases ([UO92, CW91] applicable.

One way to increase the success of the method is focusing on specific domains of applications, *e.g.*, distributed information systems. There, a couple of users cooperatively work on a set of tasks. Since objects are shared and passed between the subsystems it can be expected that different people work on the same set of objects – specified by a query. For example, each user may want to see the patients leaving the hospital next week. The first user asking this query triggers the normal evaluation. A control component ("trader") memorizes the query and the location of the answer (the view). A new query is then checked for subsumption against such views. Such an environment is currently set up in a quality management project [JJS93] where autonomous data-intensive tools cover certain aspects of quality management in the industrial product life cycle. Since the trader manipulates schema and query descriptions it provides an excellent test bed for the application of the techniques presented in this paper.

There are a couple of open questions with our approach. *First*, the issue of complex answers has not been addressed. In our model, answers are just sets of object identifiers without any derived answer attributes. These attributes are needed by application programs, and by permutation of parameters they entail additional subsumptions between queries. *Second*, it is an open problem what is the best way to evaluate the query against the subsuming view. We are interested in a minimal filter query which intersected with the view results exactly in the subsumed query. Then, it would be sufficient to test the answer candidates for satisfaction of the filter conditions.

The actual performance gain by exploiting subsumptions to views has to be validated in practical experiments. We expect good results since the clean part has been designed to capture frequently used constructs like path expressions. One should also note that the syntax of queries in Section 2 gives a view definition for free: its clean part! The first evaluation of the view creates no significant overhead since it is part of the evaluation of the original query. Afterwards, such views can be used to optimize subsequent queries.

A prototypical implementation of the proposed optimization technique is planned within the ConceptBase system. The calculus of Section 4 can serve as a starting point for developing an efficient subsumption tester for query and view concepts. This module has then to be embedded into the larger context of query modules of ConceptBase. The maintenance of the views will not create much additional work since it can re-use modules for deductive integrity checking already present in ConceptBase.

Acknowledgements. We would like to thank Maurizio Lenzerini and Matthias Jarke for fruitful discussions that contributed much to the contents of this paper. We also thank Alex Borgida and Ron Brachman for commenting on earlier drafts.

References

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, pages 159–173, Portland, Oregon, 1989.

[ASU79] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM Transactions on Database Systems*, 4(4):435–454, 1979.

[BCD92] F. Bancilhon, S. Cluet, and C. Delobel. A query language for O_2. In F. Bancilhon, C. Delobel, and P. Kanellakis, editors, *Building and Object-oriented Database System – The Story of O_2*, pages 234–255. Morgan Kaufmann Publ., 1992.

[BHHM91] H. Boley, P. Hanschke, K. Hinkelmann, and M. Meyer. COLAB: A hybrid knowledge compilation laboratory. In *3rd Int. Workshop on Data, Expert Knowledge and Decisions*, Reisensburg, Germany, September 1991.

[CGM88] U.S. Chakravarthy, J. Grant, and J. Minker. Foundations of semantic query optimization for deductive databases. In J. Minker, editor, *Foundations of deductive databases and Logic programming*, pages 243–273. Morgan Kaufmann, 1988.

[CGM90] U.S. Chakravarthy, J. Grant, and J. Minker. Logic based approach to semantic query optimization. *ACM Trans. on Database Systems*, 5(2):162–207, June 1990.

[Cha92] E. P. F. Chan. Containment and minimization of positive conjunctive queries in OODB's. In *Proc. of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 202–211, San Diego, CA, 1992.

[CW91] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proc. 17th Int. Conf. on Very Large Databases*, pages 577–589, 1991.

[DLNN91a] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91*, pages 151–162. Morgan Kaufmann, 1991.

[DLNN91b] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, Sydney, 1991.

[Gaa92] T. Gaasterland. *Generating cooperative answers in deductive databases.* PhD thesis, University of Maryland, Dept. of Computer Science, College Park, 1992.

[GL92] T. Gaasterland and J. Lobo. Processing negation and disjunction in logic programs through integrity constraints. Technical report, University of Maryland, 1992.

[Han87] E.N. Hanson. A performance analysis of view materialization strategies. In *Proc. 1987 ACM SIGMOD Int. Conf. on Management of Data*, pages 440–453, San Francisco, CA, May 1987.

[HLO91] H.H.Pang, H. Lu, and B.C. Ooi. An efficient semantic query optimization algorithm. In *Proc. Int. Conf. on DataEngineering*, pages 326–335, 1991.

[HZ80] M. Hammer and S.B. Zdonik. Knowledge based query processing. In *Proc. 6th VLDB Conf.*, pages 137–147, Montreal, Canada, 1980.

[Jar84] M. Jarke. External semantic query simplification: A graph theoretic approach and its implementation in prolog. In *Proc. 1st Int. Conf. on Expert Database Systems*, pages 467–482, Kiawah Isl., SC, 1984.

[JJS93] M. Jarke, M. A. Jeusfeld, and P. Szczurko. Three aspects of intelligent cooperation in the quality life cycle. In *Proc. Intl. Conf. Intelligent and Cooperative Information Systems*, Rotterdam, Netherlands, 1993. Keynote talk.

[JK83] D. S. Johnson and A. Klug. Optimizing conjunctive queries that contain untyped variables. *SIAM Journal of Computation*, 12(4):616–640, 1983.

[JS93] M. Jeusfeld and Martin Staudt. Query optimization in deductive object bases. In Freytag, Maier, and Vossen, editors, *Query Processing for Advanced Database Systems*. Morgan Kaufmann, June 1993.

[Kin81] J.J. King. QUIST: A system for semantic query optimization in relational databases. In *Proc. 7th VLDB Conf.*, pages 510–517, 1981.

[KKM91] A. Kemper, C. Kilger, and G. Moerkotte. Function materialization in object bases. In *Proc. 1991 ACM SIGMOD Int. Conf. on Management of Data*, pages 258–267, Denver, Colorado, May 1991.

[KKS92] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proc. 1992 ACM SIGMOD Int. Conf. on Management of Data*, pages 393–402, San Diego, CA, June 1992.

[KMWZ91] A. Kemper, G. Moerkotte, H.D. Walter, and A. Zachmann. GOM: a strongly typed, persistent object model with polymorphism. In *Proc. of BTW*, pages 198–217, Kaiserlautern, Germany, March 1991. Springer-Verlag, IFB 270.

[Kow92] W.L. Kowarschick. Semantic optimization: What are disjunctive residues useful for ? *SIGMOD RECORD*, 21(3):26–32, September 1992.

[LRS92] C. Laasch, C. Rich, and H.-J. Schek. Cocoon and Krisys– a survey and comparison. Technical Report 10/92, FB Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1992.

[LS91] M. Lenzerini and A. Schaerf. Concept languages as query languages. In *Proc. of AAAI-91*, pages 471–476, Anaheim, CA, 1991.

[OHMS92] J. Orenstein, S. Haradhvala, B. Margulies, and D. Sakahara. Query processing in the ObjectStore database system. In *Proc. 1992 ACM SIGMOD Int. Conf. on Management of Data*, pages 403–412, San Diego, CA, June 1992.

[OT92] O_2-Team. A technical overview of the O_2 system. In P. Loucopoulos and R. Zicari, editors, *Conceptual Modeling, Databases, and CASE*, pages 337–356. John Wiley & Sons, 1992.

[SK91] M. Stonebraker and G. Kemnitz. The Postgres next-generation database system. *Communications of the ACM*, 34(10):78–93, October 1991.

[SLT91] M.H. Scholl, C. Laasch, and M. Tresch. Updatable views in object oriented databases. In *Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases*, pages 189–207, Munich, Germany, December 1991. Springer LNCS 566.

[SNJ93] M. Staudt, H.W. Nissen, and M.A. Jeusfeld. Query by rule, class and concept. *Applied Intelligence*, 1993. Special issue on knowledge base management, to appear.

[SO89] S.T. Shenoy and Z.M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. on Knowledge and Data Engineering*, 1(3):344–361, September 1989.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume 2: The New Technologies*. Computer Science Press, Rockville, MD, 1989.

[UO92] T. Urpi and A. Olive. A method for change computation in deductive databases. In *Proc. 18th VLDB Conf.*, pages 225–237, Vancouver,British Columbia, Canada, 1992.

[WS92] W. A. Woods and J. G. Schmolze. The KL-ONE family. In F.W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.